Adaptive low-level control of autonomous underwater vehicles using deep reinforcement learning

Adaptive low-level control of autonomous underwater vehicles using deep reinforcement learning

Ignacio Carlucho[a,b], Mariano De Paula[a1], Sen Wang[b], Yvan Petillot[b], Gerardo G. Acosta[a]

[a]INTELYMEC group, Centro de Investigaciones en Física e Ingeniería del Centro CIFICEN – UNICEN – CICpBA – CONICET, Argentina

[b]School of Engineering & Physical Sciences Heriot-Watt University, EH14 4AS, Edinburgh, UK

ignacio.carlucho@fio.unicen.edu.ar,mariano.depaula@fio.unicen.edu.ar, s.wang@hw.ac.uk, y.r.petillot@hw.ac.uk, ggacosta@fio.unicen.edu.ar

Abstract

Low-level control of autonomous underwater vehicles (AUVs) has been extensively addressed by classical control techniques. However, the variable operating conditions and hostile environments faced by AUVs have driven researchers towards the formulation of adaptive control approaches. The reinforcement learning (RL) paradigm is a powerful framework which has been applied in different formulations of adaptive control strategies for AUVs. However, the limitations of RL approaches have lead towards the emergence of deep reinforcement learning which has become an attractive and promising framework for developing real adaptive control strategies to solve complex control problems for autonomous systems. However, most of the existing applications of deep RL use video images to train the decision making artificial agent but obtaining camera images only for an AUV control purpose could be costly in terms of energy consumption. Moreover, the rewards are not easily obtained directly from the video frames. In this work we develop a deep reinforcement learning framework for adaptive control applications of AUVs based on an actor-critic goal-oriented deep RL architecture, which takes the available raw sensory information as input and as output the continuous control actions which are the low-level commands for the AUV's thrusters. Experiments on a real AUV demonstrate the applicability of the stated deep reinforcement learning approach for an autonomous robot control problem.

Keywords: Autonomous robot; Deep Reinforcement Learning; AUV; Adaptive low-level control.

[1] Corresponding author

1. Introduction

Autonomous underwater vehicles are revolutionizing the oceanic research with applications on a vast number of scientific fields such as marine geoscience, biology and archeology but also in the private sector such as the oil and gas industry [1–3]. Over the years, there have been intensive efforts toward the development of autonomous control strategies for AUVs [4,5]. Autonomy implies that an entity can act independently according to its own criterion and it is an essential feature for engineering systems in large and uncertain environments [6]. In this sense, adaptive low-level control techniques have arisen as a way to provide autonomy to AUVs allowing them to operate in hostile environments [7].

Classical control theory [8] has evolved in a variety of methods for low-level AUV control. Several versions of the well-known PID controller [9] have been developed and used for AUV control. To name a few, in the early work of Jalving [10] a simple proportional derivative controller was proposed for AUV steering control. Fjellstad and Fossen [11] designed a PID controller for position and attitude tracking of an AUV and the global convergence of their proposal was proven by Barbalat's lemma. More sophisticated proposals can be found in the work of Valenciaga, et al. [12] where a proportional integrative controller for multiple inputs and multiple outputs (PI-MIMO) was formulated to command the rudder and the propeller of an AUV. In the work of Sutarto and Budiyono [13] a linear parameter varying (LPV) control strategy based on linear fractional transformation to formulate a robust gain schedule strategy for robust longitudinal control of an AUV was developed. To deal with the AUV modeling uncertainties and the saturations of the control actions imposed by the AUV actuators, Sarhadi et al. [14], proposed an adaptive PID formulations with anti-windup compensators [15,16] and then the stability was analyzed by Lyapunov theory and the proposed control technique was implemented in an onboard computer to be checked in a real-time dynamic simulation environment [17].

When model estimation accuracy could be imprecise and the system nonlinearities are considered, Lyapunov-based algorithms have many advantages for control formulations. An example can be found in Ferreira et al. [18] where several independent controllers have been developed, based only on Lyapunov theory, to perform decoupled motions of an AUV. In the work of Lapierre and Jouvencel [19] a nonlinear robust control formulation resorting to Lyapunov-based techniques was presented. In this case a virtual target principle was used to design an asymptotically convergent kinematic control, relying on a switching control strategy for the dynamic parameters. However, the disturbance rejection was not explicitly addressed in the formulation and the authors have explicitly recognized that further research is needed. In another way, developments coming from nonlinear control designs have been made where linear transformations were used to solve Linear Quadratic and Gaussian regulators (LQR and LQG, respectively) as in the work of Wadoo et al. [20] where a system linearization is carried out for the control of a the kinematic model of an AUV and then a LQG was formulated as a H-2 optimization problem. Geranmher et al. [21] considered a general fully coupled AUV and applied nonlinear suboptimal control, where the state-dependent Riccati equation was used to generate a suboptimal path solution. In the work of Fischer et al. [22] a continuous

robust integral of the sign of the error control was used to compensate for uncertain, nonautonomous disturbances for a coupled and fully-actuated underwater vehicle. Moreover, semiglobal asymptotic stability was proven by a Lyapunov-based stability analysis.

Underwater vehicle hydrodynamics are highly non-linear with uncertainties that are difficult to parameterize and, in addition, unknown disturbances are usually present as are typical of aquatic environments. For these reasons, researchers have resorted to adaptive controllers and have often included the dynamical model or have estimated the system parameters in the formulation of the controllers. Early, Fossen and Fjellstad [11] discussed the performance of the adaptive control laws [23,24] for controlling underwater vehicles. Afterward, several adaptive PID formulations have been proposed as in works of Antonelli et al. [25,26] where different adaptive versions based on PID control laws were formulated with an adaptive compensation of the dynamics. However, in such proposals the control gains must be adjusted manually, first in simulation and then with the real system during its operation [27]. An adaptive on-line tuning method for a coupled two-loop proportional controller of four degrees-of-freedom for an autonomous underwater vehicle is presented in the work of Barbalata et al. [28] where the gains of each controller are determined on-line according to the error signals. Rout and Subudhi [29] developed an adaptive tuning method for a PID controller using an inverse optimal control technique based on a NARMAX model [30] for the representations of the non-linear dynamics. Other adaptive feedback controller was proposed by Narasimhan and Singh [31] using LQR theory for the computation of the optimum feedback gain vector of the control system, in this case used for depth control of a low-speed underwater vehicle. These facts evidence a growing need for self-adapting controllers to environmental conditions.

To enhance the different control formulations researchers have turned their attention to artificial intelligence techniques to be incorporated in adaptive control formulations to develop real autonomous systems. Particularly, using artificial neural networks (ANNs) [32] in AUV control formulations has the advantage that the dynamics of the AUVs do not need be fully known and ANNs can learn a full, or partial, model of the nonlinear dynamics which can in turn be used for the controller design [33]. In Shi et al. [34] a hybrid control approach for AUV depth control has been proposed using the Lyapunov theory approach for the synthesis of an adaptive controller and an ANN was employed to model the depth dynamics. A dual closed loop control system was proposed in [35] where a bio-inspired model for velocity control was used in an inner control loop and a slide model controller was used in an outer tracking control loop which managed the position and orientation of an AUV. Also, a traditional Lyapunov stability analysis was carried out based on the AUV dynamic model. However, strong nonlinearities, as in underwater vehicles applications, make this analysis difficult. In this sense, after the development of the fuzzy logic [36] many fuzzy control strategies were proposed for AUV control [37–40]. Briefly, fuzzy logic control makes a smooth approximation of a nonlinear system using a fuzzy inference system [41] consisting of a set of linguistic rules about the system behavior and membership functions which must be conveniently defined. In the work of Raeisy et al. [42] a simple fuzzy control formulation can be found with two fuzzy control loops, one that controlled the roll and yaw and the other the depth of the AUV, while incorporating an optimization

procedure for the fuzzy parameters using the root mean square error between the input and the output as cost function. Recently, Khodayari et al. [43] have proposed a self-adaptive fuzzy PID controller for the attitude control of an AUV based on its previously obtained dynamic model from mechanical principles. Also, fuzzy control formulations for underwater vehicle-manipulator system (UVMS) were formulated in Esfahani et al. [44,45]. However, one disadvantage for using fuzzy control systems for AUVs is that subjective knowledge is required for the definition of the fuzzy rules and membership functions.

Other important branch with growing importance in the field of artificial intelligence for autonomous control systems is the reinforcement learning (RL) paradigm [46]. Instead of supervised learning as ANNs, RL is a mixed approach between supervised and unsupervised learning using actor-critic approach with potential advantages for adaptive control formulations in robotics [47–50]. In a nutshell, RL algorithms are able to learn a control policy through the interactions between the system and its environment. Reinforcement learning algorithms can be formulated as model-free and/or model-based [51,52]. The former uses the experience from interaction to determine directly the optimal control policy [46,53] while the latter uses it to learn/update the current model of the system or to improve the value function and/or the policy directly [54].

Particularly, for AUVs relevant works have been developed using RL formulations. In the early work of Gaskett et al. [55] a model-free RL algorithm was developed to control the thrusters responses of an AUV. More recently, Carreras et al. [56] proposed a hybrid behavior-based scheme using RL for high-level control of an AUV. In this work a semi-online neural-Q-learning algorithm was formulated using a multilayer neural network to learn the internal continuous state-action mapping of each behavior. In the work of El-Fakdi et al. [57] an on-line direct policy search algorithm based on a stochastic gradient descent method with respect to the policy parameter space was proposed. In this formulation, the policy was represented by a neural network, where its weights were the policy parameters. The states of the systems were the inputs to the neural network and the outputs were the action selection probabilities [58]. Then, El-Fakdi and Carreras [59] developed a simulation-based actor-critic algorithm using policy gradient method to solve a cable tracking task. In this formulation an initial policy is learned off-line using a hydrodynamic model of the AUV. Similarly, a two layered control architecture was proposed in [60], where an on-line RL algorithm selects the desired direction of the velocity of a marine vehicle and which, in turn, are the downstream references for a low-level proportional-derivative controller. In this work, only simulation results were reported using a computational dynamic model. In the work of Frost and Lane [61] an evaluative simulation analysis of the performance of the Q-learning algorithm for an AUV in search and inspect missions was performed using a discretized version of a continuous simulation environment to turn the problem into a grid-world type scenario. This study concluded in the need of improvements for the function approximation of the state space. In Frost et al. [62] a behavior-based architecture for AUV path planning using an actor-critic RL approach was developed. The proposed architecture regulates a set of weights of a behavior based module which, in turn, sets the control signals of the thrusters. Also, the adaptation capability of the propose approach was analyzed by a thruster failure-tolerant study for different fault scenarios. Cui et al. [63]

proposed an adaptive trajectory tracking control for AUVs using a discrete dynamical model of the underwater vehicle integrated with two artificial neural network of radial basis functions [32], one of them used to evaluate the long-time performance of the designed AUV control and the other is used to compensate the unknown dynamics. The weights of the ANN are adjusted by a standard formulation of a RL algorithm.

One of the major obstacles for RL formulations resides in dealing with applications in continuous state/action spaces when the use of function approximators is required to approximate the control policy and the state/action value functions [64,65]. Often, linear approximators are not suitable for complex systems and then nonlinear function approximators, like artificial neural networks, are required. However, the nonlinearity in ANNs may cause instabilities in the RL algorithms or may even diverge. From the developments of training algorithms for deep neural networks [66,67], Mnih et al. [68] introduced the deep Q-Network (DQN) which uses deep neural networks, i.e. convolutional neural network (CNN), to approximate the action-value function and have showed that the training of the Q function has been stabilized using experience replay and a target network. From this seminal contribution, deep reinforcement learning has emerged as a modern research field and it has become an attractive and promising framework for developing real-time adaptive control strategies to formulate adaptive control proposals for autonomous systems However, the DQN algorithm can only be applied to discrete problems, that is, with finite discretized spaces of states and actions. Llicrap et al. [69] extended deep reinforcement learning formulations for continuous state/action domains for what they developed the deep deterministic policy gradient (DDPG) algorithm based on the deterministic policy gradient (DPG) algorithm [70] incorporating the ideas of batch normalization [71] and experience replay as in [68].

Mostly, the proposed deep reinforcement learning algorithms have been tested on simulated systems mainly using simulation environments as video games simulators. Yu et al. [72] implemented the DQN algorithm to learn to avoid obstacles by learning the turning actions for a simulated car using the raw video frame images as inputs, which are directly obtained from a video game simulator. Ganesh et al. [73] used TensorFlow [74] and Keras [75] software frameworks to train a fully-connected deep neural networks, as deep RL agent, to autonomously drive across a diverse range of track geometries using a 3D car racing simulator called TORCS (The Open Racing Car Simulator) which is a modern open source simulation platform used for research in control systems and autonomous driving [76]. Similarly, El Sallab et al. [77] proposed a deep learning algorithm for autonomous driving, incorporating recurrent neural networks [32] and attention models to integrate the information and to focus on relevant information, respectively. This proposal was tested in TORCS with successful results and a good computational performance, which is an important feature for potential deployments on real robots. Specifically, for control applications of AUVs,Yu et al. [78] have solved, in a simulation environment, the trajectory tracking control problem of an AUV using a deep reinforcement learning algorithm with two embedded neural networks, the actor deep neural network and the critic deep neural network. In the formulation, the DPG algorithm was used to update the critic function and the first-order gradient-based stochastic optimization method was used to update the weights of the actor function [79].

Particularly, during the literature review a non-significant amount of previous works in deep RL has been identified for continuous control applications and, even less so, to develop autonomous control strategies for underwater vehicles. In this work we propose a deep reinforcement learning formulation with a deterministic actor-critic architecture, mainly based on the DDPG algorithm [69], adapted for low-level control of an AUV using only its on-board sensors as perception system which, in turn, becomes the inputs for the control algorithm. The successful results obtained from real experiments using an underwater vehicle demonstrated the applicability of deep RL for robotics. In this way, the obtained results demonstrated the feasibility for deep reinforcement learning to be applied on a real robot and also the encouraging results open a new promising avenue for the application of the deep reinforcement learning paradigm in the engineering community and, specifically, to develop autonomous systems into the robotics field, such as AUVs.

The paper is structured as follows: In Section 2, we briefly introduce the necessary background on reinforcement learning and the standard Q-learning algorithm as well as an overview to deep neural networks. In Section 3 we develop our proposed deep RL framework for AUV control. In Section 4, we provide experimental evidence of our proposal. Section 5 concludes the paper with the relevant contributions.

2. Background

In this section we give a non-exhaustive overview of the fundamentals of reinforcement learning as well as the well-known Q-learning algorithm that form the basis for the subsequent deep reinforcement learning developments. Following, the deterministic policy gradient method for RL formulations is summarized. Also, a brief and general overview of deep neural networks is presented which will be used as function approximators in deep RL formulations. These concepts are the basis for our proposed adaptive scheme for the low-level control of an underwater vehicle which will be developed in the next sections.

2.1. Reinforcement learning statement

The reinforcement learning problem [46] consists in learning iteratively how to achieve a goal, or to accomplish a control task, from ongoing interactions with a real or simulated system. Commonly, in RL formulations the control problem is defined by four elements, namely, the state space $\mathbb{X}$, the action space $\mathbb{U}$, the state transition probability $\mathcal{P}$ and the reward function $r_w(\cdot)$.

In a control problem, at time t, an action is a vector, $\mathbf{u}_t$, of selected values for the manipulated variables which could be the inputs to the system actuators. During the learning process, an artificial agent interacts with the system by taking an action, in our case, a new set of control actions $\mathbf{u}_t \in \mathcal{U} \subseteq \mathbb{R}^{n_u}$ and, after that, the system evolves from the state $\mathbf{x}_t \in \mathcal{X} \subseteq \mathbb{R}^{n_x}$ to $\mathbf{x}_{t+1}$ and the agent receives a numerical signal r_t called reward (or punishment) which provides a measure of how good (or bad) the action taken at $\mathbf{x}_t$ was in terms of the observed state transition. Rewards are given as hints regarding goal achievement or optimal behavior. Thus, the objective of the RL methods is to obtain the optimal policy π^* satisfying the Eq.(2), where J_π is

the expected total reward under the control policy π. The main objective of an RL agent is to learn an optimal policy, π^*, which defines the optimal control actions ($\mathbf{u}_t$) for different system's states ($\mathbf{x}_t$), bearing in mind both short and long term rewards.

$$J^* = \max_{\pi} J_\pi = \max_{\pi} \mathrm{E}_\pi\{R_t \,|\, \mathbf{x}_t = \mathbf{x}\} \tag{2}$$

Let's assume that under a given policy π, the expected cumulative reward $V^\pi(\mathbf{x})$, or value function over a certain time interval, is a function of $\mathbf{x}^\pi$, where $\mathbf{x}^\pi = \{\mathbf{x}_t\}_{t=1}^{t=n}$ are the corresponding state values and $\mathbf{u}^\pi = \{\mathbf{u}_t\}_{t=1}^{t=n}$ defines the policy-specific sequence of the agent's actions. The sequence $\mathbf{x}^\pi$ of state transitions gives rise to rewards $\{r_t\}_{t=1}^{t=n}$. Robot control is a continuous task without a single final state therefore the discounted sum of future rewards $R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$ is used to define the (discounted) expected state-value function for a policy π from the state $\mathbf{x}$, as:

$$V^\pi(\mathbf{x}) = E_\pi\{R_t \,|\, \mathbf{x}_t = \mathbf{x}\} = E_\pi\left\{\sum_{k=0}^{n} \gamma^k r_{t+k+1} \,\middle|\, \mathbf{x}_t = \mathbf{x}\right\} \tag{3}$$

where $\gamma \in (0,1]$ is the discount factor which weights future rewards. Similarly, the state-action value function is defined as:

$$Q^\pi(\mathbf{x},\mathbf{u}) = E_\pi\{R_t \,|\, \mathbf{x}_t = \mathbf{x}, \mathbf{u}_t = \mathbf{u}\} = E_\pi\left\{\sum_{k=0}^{n} \gamma^k r_{t+k+1} \,\middle|\, \mathbf{x}_t = \mathbf{x}, \mathbf{u}_t = \mathbf{u}\right\} \tag{4}$$

When the agent starts in state $\mathbf{x}$ and executes the optimal policy π^*, $V^*(\mathbf{x})$ is used to denote the maximum discounted obtained reward. Thus, the associated optimal state-value function that satisfies the Bellman's equation for all state $\mathbf{x}$ is:

$$V^*(\mathbf{x}_t) = \arg\max_{\mathbf{u}} \{r_t + \gamma . E_{\mathbf{x}_{t+1}}[(V^*(\mathbf{x}_{t+1}) \,|\, \mathbf{x}_t, \mathbf{u}_t)]\} \tag{5}$$

where $\mathbf{u}_t = \pi^*(\mathbf{x}_t)$. Similarly, the optimal state–action value function Q^* is defined by:

$$Q^*(\mathbf{x}_t, \mathbf{u}_t) = r_t + \gamma . E_{\mathbf{x}_{t+1}}[(V^*(\mathbf{x}_{t+1}) \,|\, \mathbf{x}_t, \mathbf{u}_t)] \tag{6}$$

such that $V^*(\mathbf{x}) = \max_{\mathbf{u}} Q^*(\mathbf{x},\mathbf{u})$ for all $\mathbf{x}$. Once Q^* is known through interactions, then the optimal policy can be obtained directly through:

$$\pi^*(\mathbf{x}) = \arg\max_{\mathbf{k}} Q^*(\mathbf{x},\mathbf{u}) \tag{7}$$

2.2. RL in continuous domain: AUV low-level control

The previously exposed *Q* learning method results in an adaptive control algorithm that converges on-line to the optimal control solution for completely unknown systems [46]. That is, the recursive Bellman equation (6) is solved, using data coming from system interactions without any previous knowledge of the system dynamics, to learn an optimal control policy. Commonly, in a *Q*-learning application a state-action discretization is made in advance. However, if a coarse discretization is made the results could be poor or if the discretization is too thin the *Q*-learning algorithm could become intractable. In addition, directly applying this method to a continuous control formulation, such as underwater vehicle manipulation, may be almost impracticable.

For continuous reinforcement learning, policy gradient methods are among the most widely used. These model-free methods can be applied to solve robotics problems without the need of prior knowledge of the problem or the robot dynamics. The core idea of the policy gradient methods is to improve the performance of a control policy, or simply policy, by updating the parameters of the policy function in the direction of a performance gradient. Commonly, these methods approximate a stochastic policy using an independent function approximator with its own parameters $\boldsymbol{\theta}$ that maximizes the future expected reward. However, in our formulation we use a deterministic policy gradient algorithm which has shown to be more computationally efficient than the stochastic one [70]. Thus, let $\mu(\cdot)$ be the policy function that uniquely maps states to actions, such that $\mathbf{u} = \mu_{\boldsymbol{\theta}}(\mathbf{x})$ and it has ℓ parameters grouped in a vector $\boldsymbol{\theta}$, such that $\boldsymbol{\theta} = (\theta_1, \ldots, \theta_\ell)$. Note, that at each moment that we interact with the system, we have an action vector $\mathbf{u}_t = \mu_{\boldsymbol{\theta}}(\mathbf{x}_t)$, but to simplify the notation we omit the subscript t.

Greedy policy improvements may be problematic due to the large computational load required to solve the optimization problem (Eq. (7)) in a continuous domain. Therefore instead of computing Eq. (7) it is easier to "move" the policy parameters proportionally to a feasible direction of the gradient of the action value function, *Q*, i.e.:

$$\boldsymbol{\theta}^{k+1} \propto \nabla_{\boldsymbol{\theta}} Q^{\mu^k}(\mathbf{x},\mu_{\boldsymbol{\theta}}(\mathbf{x})) \tag{8}$$

However, each state proposes a different feasible direction for the policy improvement, consequently these directions must be averaged by means of an expectation taken with respect to the state distribution ρ^{μ^k},

$$\boldsymbol{\theta}^{k+1} \propto \mathrm{E}_{\mathbf{x}\sim\rho^{\mu^k}}\left[\nabla_{\boldsymbol{\theta}} Q^{\mu^k}(\mathbf{x},\mu_{\boldsymbol{\theta}}(\mathbf{x}))\right] \tag{9}$$

therefore, by optimizing with the feasible directions we have

$$\boldsymbol{\theta}^{k+1} = \boldsymbol{\theta}^{k} + \alpha \, \mathrm{E}_{\mathbf{x}\sim\rho^{\mu^k}}\left[\nabla_{\boldsymbol{\theta}} Q^{\mu^k}(\mathbf{x},\mu_{\boldsymbol{\theta}}(\mathbf{x}))\right] \tag{10}$$

where $\alpha \in \mathbb{R}$ is a positive step-size parameter. Clearly, as can be seen in Eq. (10) the chain rule may be applied, then:

$$\boldsymbol{\theta}^{k+1} = \boldsymbol{\theta}^{k} + \alpha \, \mathrm{E}_{\mathbf{x}\sim\rho^{\mu^k}}\left[\nabla_{\boldsymbol{\theta}}\mu_{\boldsymbol{\theta}}(\mathbf{x}) \, \nabla_{\mathbf{u}} Q^{\mu^k}(\mathbf{x},\mathbf{u})\big|\mathbf{u}=\mu_{\boldsymbol{\theta}}(\mathbf{x})\right] \tag{11}$$

Using the deterministic gradient theorem, which ensures the existence of the deterministic gradient policy, such that the off-policy deterministic policy gradient is given as (for further details refer to [70]):

$$\begin{aligned}\nabla_{\boldsymbol{\theta}} J(\mu_{\boldsymbol{\theta}}) &= \int_{\mathcal{X}} \rho^{\mu}(\mathbf{x}) \nabla_{\boldsymbol{\theta}}\mu_{\boldsymbol{\theta}}(\mathbf{x}) \nabla_{\mathbf{u}} Q^{\mu}(\mathbf{x},\mathbf{u}) d\mathbf{x} \mid \mathbf{u}=\mu_{\boldsymbol{\theta}}(\mathbf{x}) \\ &= \mathrm{E}_{\mathbf{x}\sim\rho^{\mu}}\left[\nabla_{\boldsymbol{\theta}}\mu_{\boldsymbol{\theta}}(\mathbf{x}) \nabla_{\mathbf{u}} Q^{\mu}(\mathbf{x},\mathbf{u}) \mid \mathbf{u}=\mu_{\boldsymbol{\theta}}(\mathbf{x})\right]\end{aligned} \tag{12}$$

then, with (11) and (12) we have the policy updating rule,

$$\boldsymbol{\theta}^{k+1} = \boldsymbol{\theta}^{k} + \alpha \nabla_{\boldsymbol{\theta}} J(\mu_{\boldsymbol{\theta}}) \tag{13}$$

2.3. Deep neural networks

Not long ago particularly for engineering applications, most of the reported applications of artificial neural networks correspond to shallow architectures with no more than 1, 2 or 3 depth levels with deeper networks showing poorer results. However, deep neural networks have recently arisen as a way to deal with large data sets for applications in classification and regression. These new neural networks structures can be used in different areas, for example to solve engineering control problem.

Deep neural networks refer to networks organized in depth architectures as in the mammal brains [80]. Particularly, convolutional neural networks (CNNs) [67] are a class of deep neural network with a general depth topology as in Fig. 1, which have been successfully used as function approximators of the value function Q in deep reinforcement learning formulations [68]. As can be seen in Fig. 1 the architecture of a CCN network is made up of one or more convolutional layers and then followed by one or more fully connected layers as in the well-known multilayer neural networks [81]. As in classical artificial neural network applications, the number of convolutional and fully connected layers, as well as their size, must be fixed before training. These magnitudes cannot be learned and are usually referred to as hyper-parameters of the network which are given in advance. Specifically, in our application, the network inputs are given by the sensory system of the autonomous underwater vehicle which will be used to learn the low-level control task of the AUV.

Commonly, the main types of layers used to build CNN architectures are: convolutional layers, activation layers, pooling layers, and fully-connected layers. Normally, there is an input layer which contains the raw data coming from the sensory system; usually this data can be of large size and can even be in two-dimensional or three-dimensional arrays as for example, 2D or 3D images. In CNNs, the convolutional layers are only connected to a small region of the preceding layer. The network parameters consist of a set of trainable filters which convolve the input by computing the dot products between their weights and the entries that they are connected. Following the constitutional layers there are activation layers applying an elementwise activation function $\sigma(\cdot)$, commonly, a rectified linear activation function is used (ReLU) such that $\sigma(x) = \max(x;0)$ leaving the size unmodified, i.e. the input and output size of the layers are equals. The pooling layers perform a down sampling operation along the input dimensions obtaining a low-dimensional feature representation which will be the input for the following layer. Finally, fully-connected layers lie at the end of the structure containing neurons that are connected to all neurons in the previous layer, in other words, this layers work in the same way as the layers of the ordinary multilayer neural networks. In summary, deep neural network architectures are structures of sequential layers able to transform a high dimensional input data into a reduced output feature and the parameters of the networks ($\boldsymbol{\theta}$) are learned in a supervised way using training algorithms as the gradient descend method, used in backpropagation algorithms. Further details about deep neural networks can be found in [81] and the references there in.

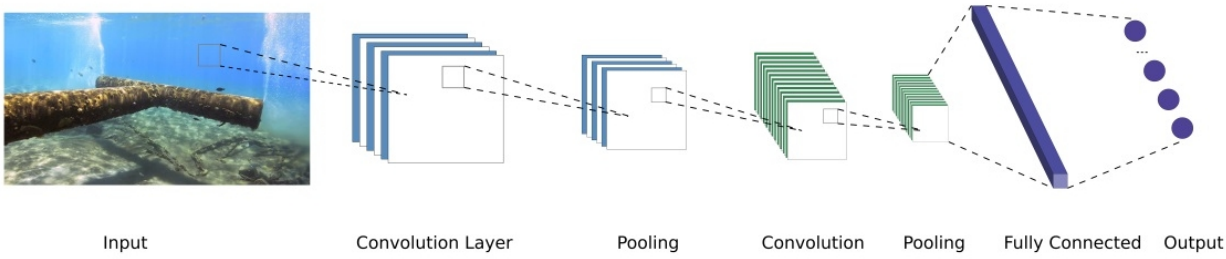


Figure 1. General deep neural network architecture.

3. Deep RL adaptive low-level control for AUV

The most common underwater vehicle configurations have four, five and even six engines. This implies that the low-level control system must simultaneously manipulate the continuous output of up to six thrusters to achieve the stated dynamic references, i.e. the set-points for the linear and angular velocities. Thus, the control system must be able to deal with a non-linear continuous problem in six degrees of freedom in an uncertain and variable environment.

Most of the deep learning control proposals have used image pixels to learn a control policy to solve complex control tasks. In addition, most of them have been tested using only simulation platforms. Also, in these cases an entire characterization of the environment is always available. However, our study aims to

propose an adaptive controller based on the previous exposed ideas for low-level control of underwater mobile robots using only the navigation measurements.

3.1. Deep RL actor-critic for continuous control

To solve the continuous control problem we employ an actor-critic model-free RL method based on the deterministic gradient theorem (Eq. (12)). In this architecture, the actor is an action selection policy that maps continuous states to continuous actions in a deterministic way and the critic is a state-value function mapping states to expected cumulative reward. However, in continuous control problems the actor and critic cannot be learned directly with the standard table-based Q-learning algorithm (Section 2) therefore function approximators are required.

In our formulation, we use a deterministic policy (as in Section 2.2) to approximate the actor behavior, $\mu_{\boldsymbol{\theta}} \sim \pi$, with parameters that are updated periodically using a recursive rule as in Eq. (13). Therefore, the adaptability is achieved by means of the continuous update of the policy parameters based on the collected experience coming from the interactions between the robot and its environment. On the other hand, the critic is approximated as $Q^{\boldsymbol{w}}(\mathbf{x},\mathbf{u}) \sim Q^{\pi}(\mathbf{x},\mathbf{u})$ with a deep network $Q^{\boldsymbol{w}}(\cdot,\cdot)$. Thus, this is a parametric function approximator, of the true state-action value function $Q^{\pi}(\cdot,\cdot)$, with all its parameters contained in a vector $\boldsymbol{w}$.

Due to the nature of the problem, the sensory system does not need to provide images to the control system and only low-level measurements of dynamic magnitudes are available at each time step t. Therefore, we use an actor-critic architecture as in Fig. 2, where deep neural networks are used for the state-action value function and policy representation, respectively. As it can be seen we used deep fully connected neural networks of ReLU layers [66] without convolutional and pooling layers for these functions approximation. In this way, we drastically simplify the network architecture [82] and also we have a compatible function approximation for the critic representation [70,83,84].

In order to learn an optimal policy, we first must obtain an optimal critic function as in Eq. (6). To do this, in a continuous domain, we consider a deep neural network as a function approximator parameterized by $\boldsymbol{w}$ and the optimal state-value function (critic function) can be found by minimizing the ordinary mean square error function, $L(\cdot)$, defined as:

$$L(\boldsymbol{w}) = \frac{1}{N}\sum_{i=1}^{N}\left(y_i - Q(\mathbf{x}_i,\mathbf{u}_i \,|\, \boldsymbol{w})\right)^2 \tag{14}$$

with gradient

$$\nabla_{\boldsymbol{w}} L(\boldsymbol{w}) = -\frac{2}{N}\sum_{i=1}^{N}\left(y_i - Q(\mathbf{x}_i,\mathbf{u}_i \,|\, \boldsymbol{w})\right)\frac{\partial Q(\mathbf{x}_i,\mathbf{u}_i \,|\, \boldsymbol{w})}{\partial \boldsymbol{w}} \tag{15}$$

where y_i are the target state-action values generated by other target deep network, $\hat{Q}$, parameterized by $\hat{\boldsymbol{w}}$, such that:

$$y_i = r(\mathbf{x}_i, \hat{\mathbf{u}}_i) + \gamma \hat{Q}(\mathbf{x}_i, \hat{\mathbf{u}}_i \,|\, \hat{\boldsymbol{w}}) \tag{16}$$

where the target action $\hat{\mathbf{u}}_i$ is given by an actor target deep network, $\hat{\mu}$, such that:

$$\hat{\mathbf{u}}_i = \hat{\mu}(\mathbf{x}_{i+1} | \hat{\boldsymbol{\theta}}) \tag{17}$$

Then, the actor policy function represented by the deep network μ, is updated determining the critic parameters $\boldsymbol{\theta}$ using the deterministic gradient theorem for optimizing the expected return (as in Eq. (12)). Thus, after the critic function is found it is used to update the actor function, being $\mathbf{u}_t = \mu(\mathbf{x}_t|\boldsymbol{\theta})$ and $J(\mu_{\boldsymbol{\theta}}) = Q(\mathbf{x}_i, \mu(\mathbf{x}_i|\boldsymbol{\theta})|\boldsymbol{w})$, the deterministic policy gradient is given as in Eq. (18) and we use a stochastic optimization method [79] to obtain the optimal policy representation.

$$\nabla_{\boldsymbol{\theta}} J = \frac{\partial Q(\mathbf{x}_i, \mu(\mathbf{x}_i|\boldsymbol{\theta})|\boldsymbol{w})}{\partial \mathbf{u}_i} \cdot \frac{\partial \mu(\mathbf{x}_t|\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} \tag{18}$$

Note that to improve the stability of the learning process separated deep networks are used for generating the Q-learning targets y_i and for the critic approximations as in [68].

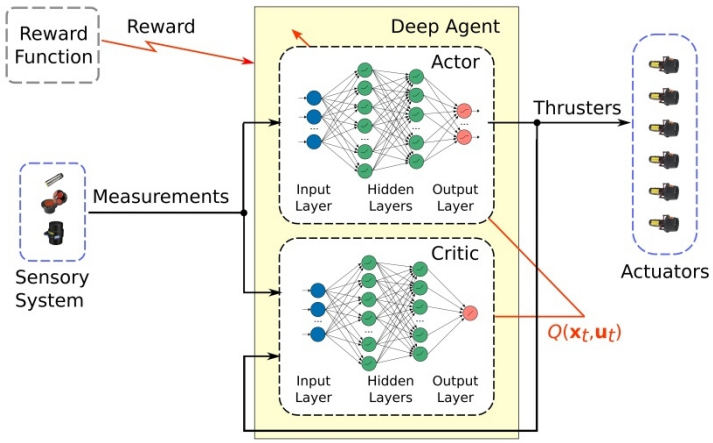


Figure 2. Actor-critic architecture using deep fully connected neural network of ReLU layers as function approximators.

3.2. Goal-oriented actor-critic control architecture

Commonly, deep reinforcement learning approaches are designed to learn a control policy that depends only on the current system state and the goal implicitly lies in the system behavior. However, every time that

the goal of the task changes it is necessary to learn the networks parameters of the actor and critic function (Fig. 2). Thus, for an autonomous control system for an underwater vehicle, this means that the deep agent of the actor-critic architecture showed in Fig. 2 must be re-trained every time that the dynamic references are substantially modified.

Particularly, for our presented deep RL formulation, it is necessary that the agent can generalize for problems that are similar in nature, but different in the required controlled action taken to solve it and this depends on the current goal the agent is required to achieve. In order to achieve an adaptive low-level control, based on the actor-critic architecture of Fig. 2, we enhanced this proposal by formulating a goal-oriented control architecture as shows Fig. 3. In this scheme, the deep agent learns a control policy based on the system dynamic information as well as the current target specification and based on information about the behavior of the autonomous control system itself, which is summarized in its last taken decision.

The system dynamic information is the set of measurements of the controlled variables, performed by the robot sensory system, which are combined in a vector $\mathbf{x}_t^c$. The current target specifications are the references given for the controlled low-level variables, combined in vector $\mathbf{x}_{ref_t}^c$. Thus, with this information, it is possible to obtain a characterization about the discrepancy between the current and the expected dynamic behavior of the robot. This information is given in an instantaneous error vector, $\mathbf{e}_t$, computed between the measurements of the controlled variables $\mathbf{x}_t^c$, at time t, and the fixed set-points for such magnitudes $\mathbf{x}_{ref_t}^c$. Thus, $\mathbf{e}_t$ provides instantaneous information to the deep agent about the performance of the autonomous control system itself. Also, the deep agent receives information about the control system behavior, summarized in the last executed action, $\mathbf{u}_{t-1}$.

As can be seen in Fig. 3, the deep agent receives information summarized in the markovian system state $\mathbf{x}_t = [\mathbf{x}_t, \mathbf{e}_t, \mathbf{u}_{t-1}]^T$. An advantage of perceiving the system state ($\mathbf{x}_t$) in this way is the fact that it only involves readily known variables, yet they are informative enough to describe the autonomous system state for a successful low-level control. In addition, the state configuration, $\mathbf{x}_t$, only contains measurable information coming from common sensors, which are widely used in the field of underwater applications, avoiding expensive computational treatments as in video images which are widely used as input in most of the deep learning applications but are not, however, mainly applied in the underwater domain.

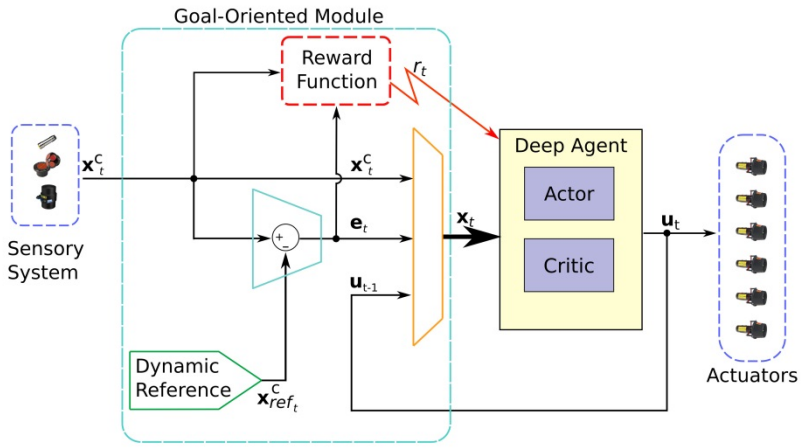


Figure 3. Goal-oriented control architecture based on actor-critic architecture.

3.3. Deep RL algorithm for AUV low-level control

Based on the presented goal-oriented actor-critic architecture, following we develop our deep RL approach for adaptive low-level control for AUVs. Therefore, an algorithmic representation will be developed aiming to present a computationally feasible version. Thus, Algorithm 1 outlines the pseudocode of our deep learning agent for underwater vehicle low-level control.

In line 1 of Algorithm 1, the inputs are the maximum number of training episodes M, the time horizon of each episode T, the maximum size of the replay buffer m^+, the minimum size of the replay buffer m_-, the number N of state transitions $\mathcal{T}_i = (\mathbf{x}_i, \mathbf{u}_i, r_i, \mathbf{x}_{i+1})$ to be taken from the replay buffer $\mathcal{R}$ to define a subset $\mathcal{S} = \{\mathcal{T}_1, ... \mathcal{T}_i, ... \mathcal{T}_N\}$ such that $\mathcal{S} \subseteq \mathcal{R}$ which will be used for minibatch training, the discount rate γ, the updating rate β for the deep target networks parameters, the reward function $r_w(\cdot)$ and the temporally correlated Ornstein-Uhlenbeck process noise with scale factor φ and mean and variance parameters ρ and ν, respectively. This process is incorporated for exploration purposes (line 11).

Algorithm 1 was formulated to learn a control task from scratch but it also can continue learning from data of a pre-trained policy represented in the deep networks $Q(\cdot, \cdot | \boldsymbol{w})$, $\mu(\cdot \mid \boldsymbol{\theta})$ and in a replay buffer $\mathcal{R}$. For this reason, in line 2 and in line 4, there are both options: to *initialize* or *load*. In line 3 the target networks, $\hat{Q}$ and $\hat{\mu}$ are initialized with the same parameterization of Q and μ, i.e. $\hat{\boldsymbol{w}} = \boldsymbol{w}$ and $\hat{\boldsymbol{\theta}} = \boldsymbol{\theta}$.

In the RL paradigm, the low-level control problem of an AUV can be seen as a continuous control task. Thus, from an algorithmic point of view, each training episode j is defined along a time horizon T. In Algorithm 1, each learning episode is carried out in the loop from line 5 to line 33. At the beginning of each episode the random Ornstein-Uhlenbeck stochastic process is initialized (line 6), in order to carry out the environment exploration and the dynamic variables of the system are obtained from the sensory system (line 7) to set the initial system state $\mathbf{x}_1$ (line 8).

Into the inner loop from line 9 to 31 the core of the deep RL low-level AUV control algorithm is performed. Keeping in mind that our proposal is developed to be applied on a real robot we must keep a fixed sample time, dt. Then, the loop execution time must strictly be as long as a sampling time, dt, to

satisfy the hardware constrains imposed by the technological the system. Thus, to achieve this requirement we use a *timer* to manage the execution time and guarantying a sample time dt. So, in line 10, we initialize a *timer* which waits a time lapse dt to continue with the execution of the Algorithm 1 in line 27. Note that dt must be long enough to allow the execution of the commands from line 11 to 26. Using the current actor control policy a control action is determined (line 11) and it is immediately sent to the actuators of the underwater vehicle (line 12).

Aiming to improve the stability of the learning process and to make an efficient use of the computational resources, we implement batch learning [85] using an experience replay buffer $\mathcal{R}$ which can reach a maximum size m^+. Thus, in $\mathcal{R}$ the experience is stored in the form of transitions $\mathcal{T}_i$, such that $\mathcal{R} = \{\mathcal{T}_1, \mathcal{T}_2 ..., \mathcal{T}_{m^+}\}$. In this way, after each interaction step, the actor and critic are updated based on the experience stored in a replay buffer $\mathcal{R}$. To do this, if the buffer has stored at least m_- transitions (i.e. condition of line 13 is true), a random minibatch $\mathcal{S}$ of experimented transitions is sampled from $\mathcal{R}$ (line 14). Then, we this subset of previous experience, into the inner loop from line 15 to 18 the state-action value targets (y_i) are computed, which are necessary to obtain the critic parameterization $\boldsymbol{w}$, by miniminzing the loss function (Eq.13), and to obtain the actor parameterization $\boldsymbol{\theta}$. In this way, the actor and critic deep networks are parameterized as $\mu(\cdot|\boldsymbol{\theta})$ and $Q(\cdot|\boldsymbol{w})$ (line 19-20). In line 21-22 the parameters of the target networks, $\hat{\mu}(\cdot|\boldsymbol{\theta})$ and $\hat{Q}(\cdot|\boldsymbol{w})$, are updated.

As was said, we use a replay buffer $\mathcal{R}$ to store the experience thus when the buffer reaches its allowable maximum size m^+ we simply remove the oldest stored experience (line 24-26). Thus, with the dynamic measurements obtained from the sensory system the transition state representation $\mathbf{x}_{t+1}$ is made (line 28) and the instantaneous reward signal is computed using the reward function, i.e. $r_t = r_w(\mathbf{x}_{t+1})$ (line 29). Next, with this information, the experimented transition $(\mathbf{x}_t, \mathbf{u}_t, r_t, \mathbf{x}_{t+1})$ is incorporated into the buffer $\mathcal{R}$ (line 30).

Finally, the outputs of the algorithm (line 34) are the low-level control policy, synthetized in the deep network $\mu(\cdot \mid \boldsymbol{\theta})$, the critic function summarized in the deep network $Q(\cdot, \cdot|\boldsymbol{w})$ and the buffer replay $\mathcal{R}$.

Algorithm 1. Deep RL algorithm for AUV low-level control

1. **Inputs**: M, T, m^+, m_-, N, γ, β, $r_w(\cdot)$,φ, ρ, ν
2. Randomly initialize/load critic network $Q(\cdot,\cdot|\boldsymbol{w})$ and actor network $\mu(\cdot\,|\,\boldsymbol{\theta})$ with weights $\boldsymbol{w}$ and $\boldsymbol{\theta}$, respectively
3. Initialize target networks $\hat{Q}$ and $\hat{\mu}$ with weights $\hat{\boldsymbol{w}} = \boldsymbol{w}$ and $\hat{\boldsymbol{\theta}} = \boldsymbol{\theta}$
4. Initialize /load replay buffer $\mathcal{R}$
5. **For** $j = 1$ to M **do**
6. Initialize a random process $\mathcal{N}_t(\varphi,\rho,\nu)$ for action exploration
7. Get AUV dynamic measurements from sensory system
8. Set initial state $\mathbf{x}_1$
9. **For** $t = 1$ to T **do**
10. Initialize the $timer$
11. Select action $\mathbf{u}_t = \mu(\mathbf{x}_t\,|\,\boldsymbol{\theta}) + \mathcal{N}_t$ according to the current policy and exploration noise
12. Execute action $\mathbf{u}_t$ over the system
13. **If** $|\mathcal{R}| > m_-$ **then**
14. Sample a random minibatch $\mathcal{S}$ of N transitions $\mathcal{T}_i$ from $\mathcal{R}$, such that $\mathcal{S} = \{\mathcal{T}_1,...\mathcal{T}_i,...\mathcal{T}_N\} \subseteq \mathcal{R}$
15. **For** $i = 1$ to N
16. With the target actor function $\hat{\mu}(\cdot\,|\hat{\boldsymbol{\theta}})$ obtain $\hat{\mathbf{u}}_{i+1} = \hat{\mu}(\mathbf{x}_{i+1}|\hat{\boldsymbol{\theta}})$ (Eq. (17))
17. Set the state-action value target $y_i = r_i + \gamma\hat{Q}(\mathbf{x}_{i+1},\hat{\mathbf{u}}_{i+1}\,|\,\hat{\boldsymbol{w}})$ (Eq.(16))
18. **End**
19. Obtain the critic parameterization $\boldsymbol{w}$ minimizing the loss function $L(\boldsymbol{w})$ (Eq.(14))
20. Obtain the actor policy parameterization $\boldsymbol{\theta}$ using the deterministic policy gradient $\nabla_{\boldsymbol{\theta}}J$ (Eq. (18))
21. Update the actor target network parameterization $\hat{\boldsymbol{\theta}} \leftarrow \beta\boldsymbol{\theta} + (1-\tau)\hat{\boldsymbol{\theta}}$
22. Update the critic target network parameterization $\hat{\mathbf{w}} \leftarrow \beta\boldsymbol{w} + (1-\tau)\hat{\mathbf{w}}$
23. **End**
24. **If** $|\mathcal{R}| > m^+$ **then**
25. Remove the oldest $\mathcal{T}_* \in \mathcal{R}$ from the replay buffer $\mathcal{R}$, i.e. $\mathcal{R} = \{\mathcal{R}\} - \{\mathcal{T}_*\}$
26. **End**
27. Wait until $timer$ is over
28. Get AUV dynamic measurements from sensory system and set the transition state $\mathbf{x}_{t+1}$
29. Observe the reward r_t, i.e. $r_t = r_w(\mathbf{x}_{t+1})$
30. Store the transition $(\mathbf{x}_t, \mathbf{u}_t, r_t, \mathbf{x}_{t+1})$ in $\mathcal{R}$
31. **End**
32. Reset time, i.e. $t = 1$
33. **End**
34. $Q(\cdot,\cdot|\boldsymbol{w})$, $\mu(\cdot\,|\,\boldsymbol{\theta})$, $\mathcal{R}$

4. AUV control experiments and results discussion

4.1. Experimental setup

In order to test our proposal the underwater vehicle Nessie VII (Fig. 4) developed by the Heriot-Watt University was used [86]. Briefly, this robot has six thrusters, indicated as T_1 to T_6 in Fig. 4, allowing for a five degree of freedom control and it is equipped with a DVL and an IMU to measure the linear and angular velocities. This underwater vehicle serves as an excellent platform for testing and development of underwater applications and it has already been used in various research articles as an experimental platform [28,61,62,87].

During the experiments the robot interacted with an external computer using ROS (Robot Operating System), exchanging messages in a network, with a sampling time $dt = 0.1$ seconds. In this way, the on-board computer managed the sensory and navigation systems, while the external computer held the RL controller. The underwater vehicle is controlled by setting a vector $\mathbf{u}_t = (u_t^1, u_t^2, u_t^3, u_t^4, u_t^5, u_t^6)$, where $u_t^1, u_t^2 \ldots u_t^6$ are the thrusters commands, at time t, for the thruster 1, 2, … 6, respectively. These commands are determined by a control policy, synthetized in the actor deep neural network $\mu(\cdot\,|\boldsymbol{\theta})$ (Algorithm 1).

In our deep RL problem formulation we define the markovian system state at time t, using the observable state variables given by the instantaneous measurements from the robot sensors, as $\mathbf{x}_t = (\mathbf{v}_t, \boldsymbol{\omega}_t, \dot{\mathbf{v}}_t, \dot{\boldsymbol{\omega}}_t, \mathbf{u}_{t-1}, \mathbf{e}_t)$. The magnitudes $\mathbf{v}_t = (v_x, v_y, v_z)$ and $\boldsymbol{\omega}_t = (\omega_x, \omega_y, \omega_z)$ are the linear and angular velocities with respect to the axes x, y and z, given by the DVL and IMU, respectively. Analogously, $\dot{\mathbf{v}}_t = (\dot{v}_x, \dot{v}_y, \dot{v}_z)$ and $\dot{\boldsymbol{\omega}}_t = (\dot{\omega}_x, \dot{\omega}_y, \dot{\omega}_z)$ are the linear and angular accelerations with respect to the axes x, y and z, respectively. While $\mathbf{u}_{t-1}$ is the vector of the commands executed in the previous time step $t-1$ and $\mathbf{e}_t$ is the instantaneous velocity error computed between the velocities at time t and the fixed set-points.

We seek to minimize the deviations of the controlled dynamic variables from their references whilst also trying to minimize the thruster use, to reduce overall energy consumption, and sudden variations of the controlled signals. Note that in order to accomplish this we propose an appropriate reward function $r_w(\cdot)$ as in Eq. (19). In this way, the immediate reward, r_t, is given by an evaluation of the effects the executed action ($\mathbf{u}_t$) had in the state of the system. This evaluation consists of three different terms:

$$r_t = \lambda \exp\left(-\frac{1}{a^2}\left(\mathbf{x}_t^{\mathrm{c}} - \mathbf{x}_{ref}^{\mathrm{c}}\right)^T \Lambda \left(\mathbf{x}_t^{\mathrm{c}} - \mathbf{x}_{ref}^{\mathrm{c}}\right)\right) - \zeta \sum_{i=1}^{6} |u_i| - \xi \left|\left|\overline{\mathbf{u}}_{t-\tau:t-1} - \mathbf{u}_t\right|\right| \tag{19}$$

where the first term evaluates the square error between the controlled dynamic variables ($\mathbf{x}_t^{\mathrm{c}}$) and their references ($\mathbf{x}_{ref}^{\mathrm{c}}$) with $\Lambda = \mathrm{diag}([\ell_1, \ell_2, \ldots, \ell_{n_c}])$ and ℓ_k, $k = 1, \ldots, n_c$, being the characteristic length-scales, the second term weights the thruster usage and the last term penalizes sudden changes in thrusters commands by computing the norm between the current action ($\mathbf{u}_t$) and the moving average of past taken actions ($\overline{\mathbf{u}}_{t-\tau:t-1}$). The mean $\overline{\mathbf{u}}_{t-\tau:t-1}$ is backward computed using a slide windows of length τ. The parameters λ, ζ and ξ

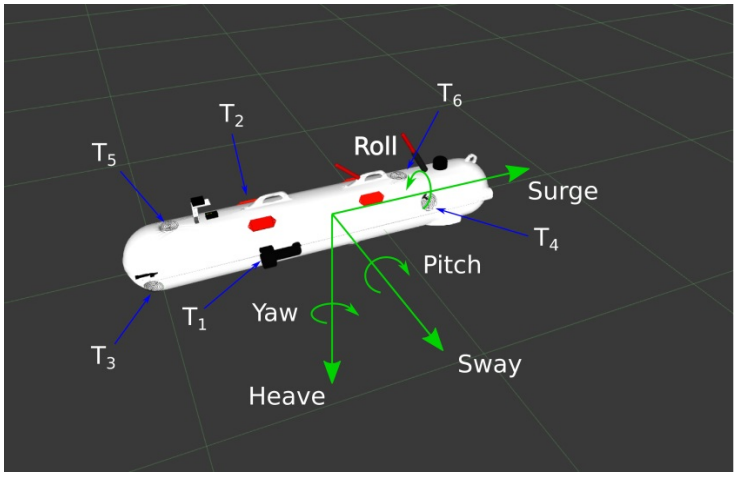
are scale factors $\in (0, 1]$. Note that the first term of Eq. (19), penalizes for great deviations of the controlled variables from their reference but saturates for significant deviations. In this manner, we aim to constrain the reward magnitude to avoid numerical instability in the learning process. Observing Eq. (19) it can be clearly seen that as the value of *a* decreases, the reward function spans a smaller interval for the controlled dynamic variables and consequently the control task is more challenging. Note that the optimal control policy (μ) should generate a sequence of actions $\mathbf{u}_1$, $\mathbf{u}_2$,…, $\mathbf{u}_t$, … such that the cumulative reward is maximized instead of maximizing each immediate reward in Eq. (19).

To illustrate the significance of each term for the reward function (Eq. (19)), Fig. 5 depicts simulated results[2], under the same training conditions, for different configurations of the reward function. Hereinafter, in all trials we fix $M = 500$ episodes of length $T = 700$ time steps with sampling time of 0.1 seconds. The minimum and maximum size for the replay buffer $\mathcal{R}$ were set in $m^- = 100$ and $m^+ = 200000$ elements. The number of state transitions for the minibatch sampling was fixed as $N = 60$. We use a discount rate $\gamma = 0.99$ and the updating rate for the deep target networks parameters is $\beta = 0.001$. Fig. 5a shows the results for a simple reward function where only the deviations of the controlled dynamic variables from their references are taken into account, i.e. $\lambda \neq 0$ and $\zeta = \xi = 0$. As can be seen, Algorithm 1 finds a policy capable of achieving the target references for the dynamic variables, however the thruster output patterns are too variable and aggressive to be applied in a real underwater vehicle. Fig. 5b depicts the results for a reward function that incorporates the second term, that is penalizing the usage of the thrusters, i.e. $\lambda \neq 0$, $\zeta \neq 0$ and $\xi = 0$ in Eq. (19). As can be seen, Algorithm 1 finds a control policy capable of successfully control the AUV but the performance of the thruster output is still not suitable to be applied on real actuators. Following, we incorporate the third term to the reward function, i.e. we use the Eq. (19) with $\lambda \neq 0$, $\zeta \neq 0$ and $\xi \neq 0$, and again performed the same training experiment with Algorithm 1, obtaining the results showed in Fig. 5c. As it can be seen, the improvements in the results are clearly noticeable which validates the use of a reward function like the one presented in Eq. (19).

The implementation of the Algorithm 1 was done in Python using Tensorflow[3], a machine learning library with specially developed tools for deep learning applications. As was mentioned in Section 3.1, for the policy network we used a deep fully connected neural network, with an input layer of size 21, three hidden layers using ReLU activation functions, of size 600, 400 and 300, and one output layer of size 6, with sigmoid activation function, giving a total of 375056 free parameters. The state-action value function uses a similar deep neural network structure, with the difference that the state vector is fed to the input layer, and the action vector is fed to the first hidden layer.

[2] For simulation we use the Nessie simulator [86].
[3] For further details refer to the web site https://www.tensorflow.org/.

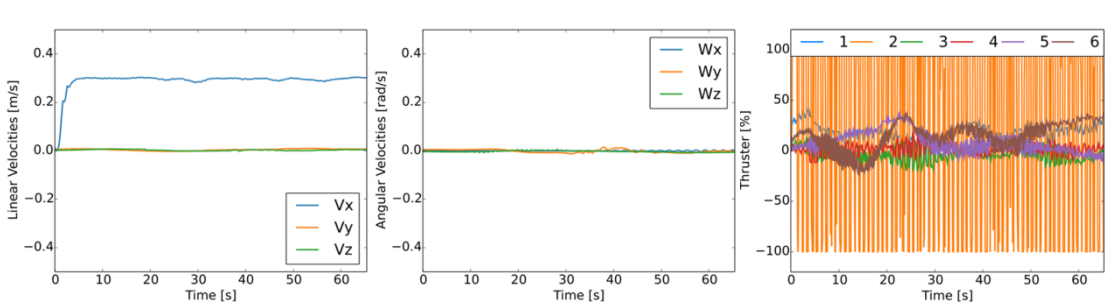
(a)



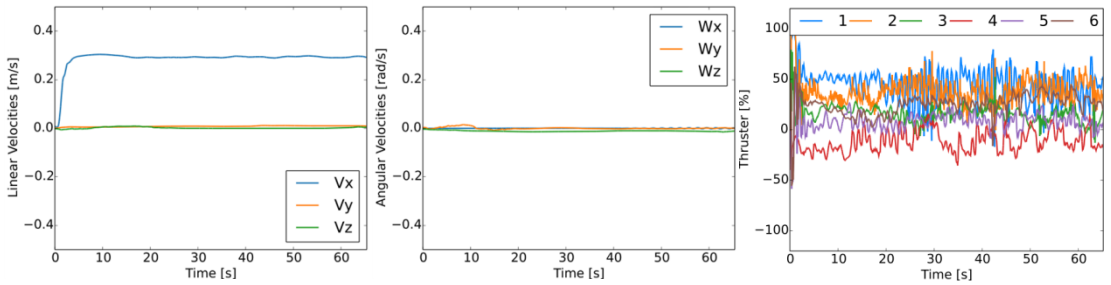
(b)

Figure 4. The autonomous underwater vehicle, Nessie VII, designed and built at the Heriot-Watt University.

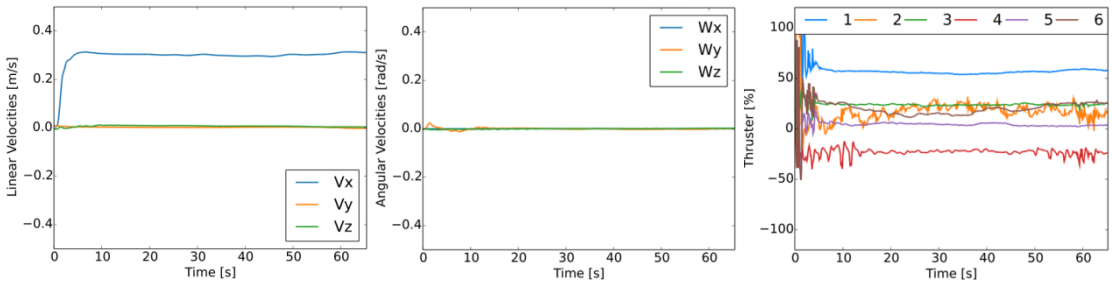
a) Experimental platform and facilities; **b**) AUV reference system.



(a)



(b)



(c)

Figure 5. Comparative examples for different structures for the reward function.

a) Results for a reward function of Eq. (19) with $\lambda \neq 0$ and $\zeta = \xi = 0$; **b**) Results for a reward function of Eq. (19) with $\lambda \neq 0$, $\zeta \neq 0$ and $\xi = 0$; **c**) Results for a reward function of Eq. (19) with $\lambda \neq 0$, $\zeta \neq 0$ and $\xi \neq 0$.

4.2. Wet experimental results

In order to test our proposed deep reinforcement learning approach for adaptive low-level control of an AUV, we used the underwater vehicle Nessie VII as an experimental platform to carry out a number of experiments. This underwater robot is equipped with all the necessary instruments for underwater navigation (compass, gyroscope, accelerometers, DVL, IMU, depth meter, and others).

As was explained in Section 3.2, Algorithm 1 was formulated so as to learn a control task from scratch but it can also continue the learning process from data of a pre-trained policy summarized in the deep networks $Q(\cdot,\cdot|\boldsymbol{w})$, $\mu(\cdot\,|\,\boldsymbol{\theta})$ and in a replay buffer $\mathcal{R}$. So, by taking advantage of this fact, we first carried our training experiments on the simulator and then, in a following training step, the learning procedure continues on-line, in the real vehicle, improving and adapting the low-level control policy. This operational scheme gives flexibility, reducing costs and operational risks by learning initially on the simulator. Besides, it demonstrates the capability of the proposed technique to self-adapt, since it is capable to overcome the inevitable behavioral gap between the simulator and the real robot in its environment.

Initially, during the training phase with the AUV simulator, in each training episode the set-points of the dynamic specifications ($\mathbf{x}_{ref}^{c}$) for the controlled variables ($\mathbf{x}_t^{c}$) are randomly chosen within a certain range of possible values. In this way, we seek to learn a control policy able to manipulate the underwater vehicle within an operation range instead of doing so only in a neighborhood of an operating point.

The experiments with the real robot were carried out in the Ocean System Laboratory of the Heriot-Watt University. Due to the physical constrains imposed by the AUV and the experimental facilities, we only controlled the lineal velocities (*surge*, *sway* and *heave)* and the rotational movements around its vertical and transversal axes (*yaw* and *pitch*, respectively). Therefore, the controlled linear velocities are limited to: $-0.5\ \mathrm{m/s} < v_x < 0.5\ \mathrm{m/s}$, $-0.2\ \mathrm{m/s} < v_y < 0.2\ \mathrm{m/s}$ and $0\ \mathrm{m/s} < v_z < 0.2\ \mathrm{m/s}$ and the angular velocities are constrained as $\omega_y = \omega_z = 0$ (no pitch or yaw velocities). Note that, v_x refers to forward/backward speed (surge), v_y is the lateral speed to the left/right (sway) and v_z is the vertical velocity (heave). Also note that the angular movement around the longitudinal axis (roll, ω_x) of the AUV is not controlled. Thus, at a certain time t, the vector of controlled variables is defined as $\mathbf{x}_t^{c} = (v_x, v_y, v_z, \omega_y, \omega_z)$. The immediate reward is computed according to Eq. (19) with $a = 1$, $\lambda = 0.75$, $\zeta = 0.1$, $\xi = 0.4$ and $\Lambda = \mathrm{diag}([1, 0.75, 0.75, 0.25, 1]$, with a given set point for the controlled variables, $\mathbf{x}_{ref}^{c} = (v_{x_r}, v_{y_r}, v_{z_r}, \omega_{y_r}, \omega_{z_r})$ and the action vectors $\mathbf{u}_t$, $\mathbf{u}_{t-1}$,…, $\mathbf{u}_{t-\tau}$ with $\tau = 100$.

Using Nessie's simulator, we ran Algorithm 1 during 500 training episodes (this is equivalent to almost 10 hours of real interaction) obtaining an initial low-level control policy. Afterwards, we continued the training phase using the real robot, fixing different set-points for each training episode. In the rest of the section we show and discuss the obtained results of applying Algorithm 1 for the low-level control of the AUV so to achieve the fixed dynamic specifications. We set different operational conditions to demonstrate the adaptive features of the proposed algorithm to adapt the control policy. It is worth noting that as the algorithm runs, the learning process is actively seeking to achieve the dynamic specifications while simultaneously improving the control policy. In this sense, it is also worth mentioning that the dynamics of the vehicle changes substantially for different operating conditions, for example its forward behavior is very different from the backward or lateral behavior.

Figure 6 shows the results of a training episode carried out on the real experimental platform (Fig. 4), with a reference $\mathbf{x}_{ref}^{c} = (v_x, v_y, v_z, \omega_y, \omega_z) = (0.4\ \mathrm{m/s}, 0, 0, 0, 0)$. As it can be seen, the control policy drives

the vehicle to the reference in just 10 seconds. Note that the oscillation of the controlled variables around its reference values are directly associated with the noise of the measuring instruments and, it can also be seen that the angular velocities measurements are noisier. The right panel of Fig. 6 shows the usage (in percentage) of each thruster.

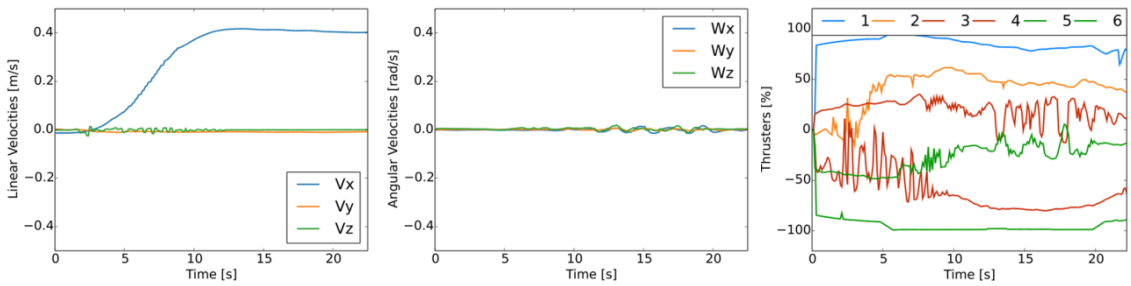


Figure 6. Results of a training episode for $\mathbf{x}_{ref}^{c} = (0.4\ \mathrm{m/s}, 0, 0, 0, 0)$ during the execution of Algorithm 1.

Figure 7 shows the results of a training episode with $\mathbf{x}_{ref}^{c} = (v_x, v_y, v_z, \omega_y, \omega_z) = (-0.4\ \mathrm{m/s}, 0, 0, 0, 0)$, in which the low-level control policy takes around ten seconds to reach the set-point. This case is useful, since it allows to analyze the capability of Algorithm 1 to adapt the low-level control policy to a different operation condition. Unlike the previous case, the learned control policy must drive the vehicle in a linear backward movement. This operation setting is different to the previous results (Fig. 6) in the sense that the forward and the backward dynamics exhibited a different behavior. This difference has multiple reasons, for example, the thrusters are not symmetrically located in the AUV body, the force made by the propellers may vary according to the direction of rotation, among others. In summary, this example further demonstrates the ability of the proposed Algorithm 1 to adapt itself while facing a different situation.

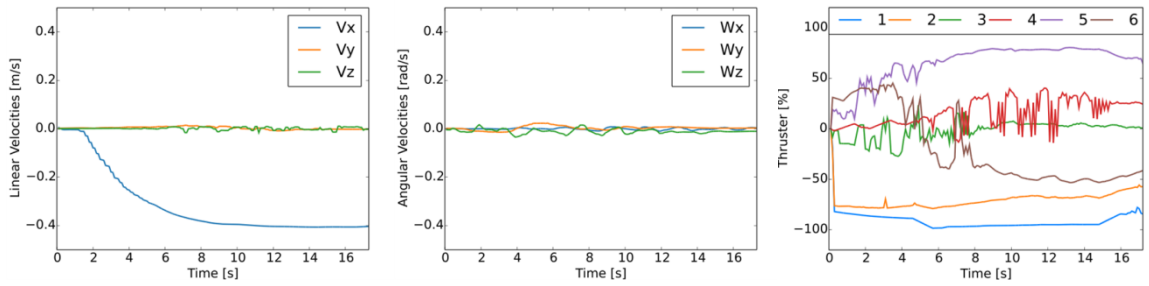


Figure 7. Results of a training episode for $\mathbf{x}_{ref}^{c} = (-0.4\ \mathrm{m/s}, 0, 0, 0, 0)$ during the execution of Algorithm 1.

In Fig. 8 the results of an episode with a positive heave velocity as set-point are shown. The requested heave velocity was set to 0.18 m/s, i.e. $\mathbf{x}_{ref}^{c} = (0, 0, 0.18\mathrm{m/s}, 0, 0)$. As it can be seen, the low-level control

policy successfully achieves the specification. The thrusters 5 and 6 are placed vertically in the AUV (Fig. 4), therefore their relative higher usage can be easily understood as it is showed in the right panel of Fig. 8.

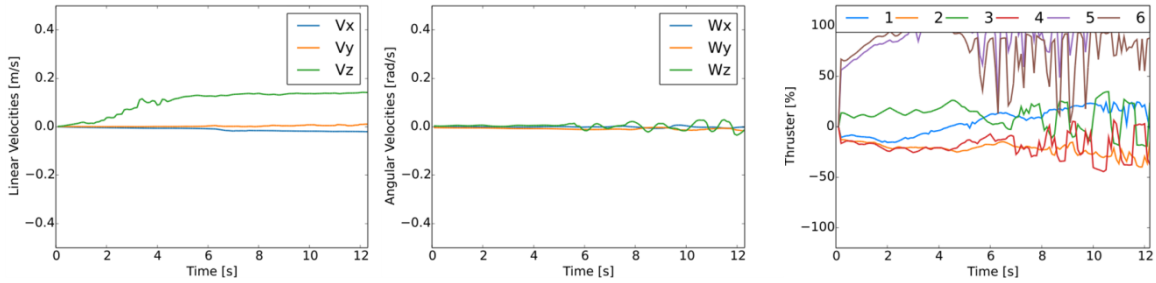


Figure 8. Results of a training episode for $\mathbf{x}_{ref}^{c} = (0{,}0,\ 0.18\ \mathrm{m/s},\ 0,\ 0)$ during the execution of Algorithm 1.

A more complex control task was imposed by setting $\mathbf{x}_{ref}^{c} = (0, -0.1\ \mathrm{m/s}, 0, 0, 0)$, where a pure lateral movement (sway) is requested. It should be noted that the thrusters of the AUV are not arranged so as to directly generate a pure lateral displacement, that is, the control system must determine the necessary thrust composition of the engines to achieve this pure lateral displacement. Figure 9 shows the obtained results for a training episode with sway velocity $v_y = -0.1$. As it can be seen, the control policy can successfully drive the vehicle towards the specified set-point in almost 16 seconds.

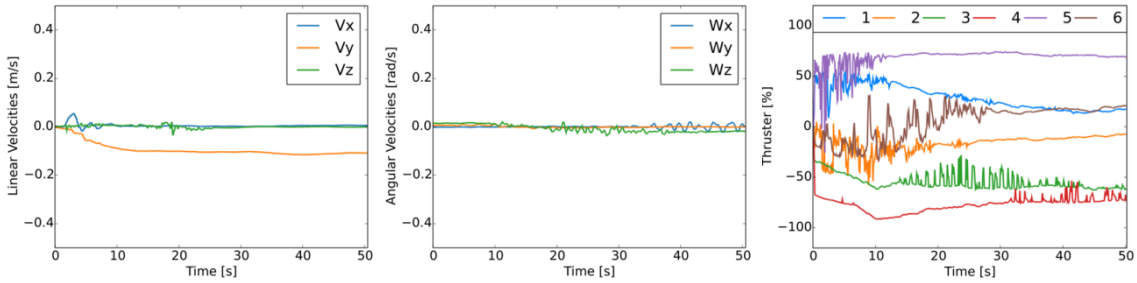


Figure 9. Results of a training episode for $\mathbf{x}_{ref}^{c} = (0, -0.1\ \mathrm{m/s}, 0, 0, 0)$ during the execution of Algorithm 1.

In order to make the control task even harder, during a training episode the set-point was set as $\mathbf{x}_{ref}^{c} = (-0.2\mathrm{m/s}, -0.1\ \mathrm{m/s}, 0, 0, 0)$. With this requested reference the control policy must achieve a complex combined movement of the AUV with a simultaneous backward and lateral motion with a speed of 0.2 m/s and 0.1 m/s, respectively. Figure 10 shows the obtained results for this episode. In this case, again, the low-level control policy was successfully adapted by Algorithm 1 achieving the requested velocities in almost 18 seconds.

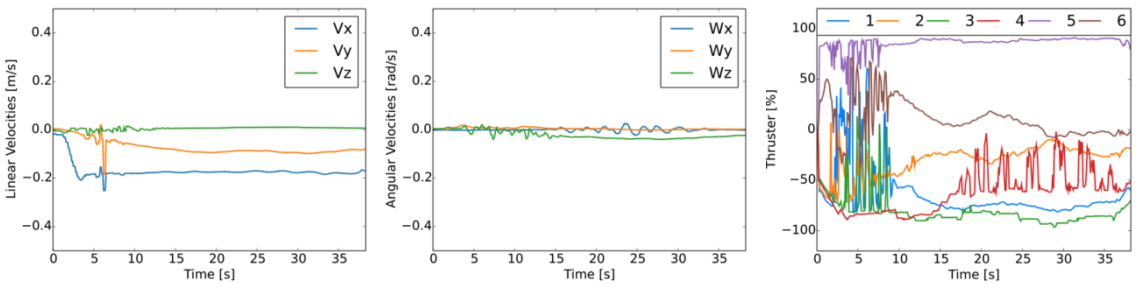


Figure 10. Results of a training episode for $\mathbf{x}_{ref}^{c} = (-0.2\text{m/s}, -0.1\ \text{m/s}, 0, 0, 0)$ during the execution of Algorithm 1.

If we look at the responses of the AUV to the executed control actions by the agent to follow a simple reference velocity, for example implying only a one-dimensional movement (as in Fig. 6), we cannot directly observe a simple pattern of thrusters' usage. This fact shows that an underwater vehicle is a dynamic system with a complex couple dynamics hard to be modeled and controlled, which justifies the development of this kind of adaptive control techniques.

5. Final Remarks

In this work an adaptive controller based on the deep reinforcement learning framework was proposed for low-level control of an AUV. The proposed algorithm uses only the low-level data provided by the on-board sensors of the vehicle to make the decisions needed for successfully solving the continuous control task. Moreover, unlike classic control theory, which requires a model of the system, or fuzzy control strategies, that requires prior expert knowledge, the proposed algorithm carries out a specialization process with minimum prior knowledge. Effectively, using only the input parameters the deep agent is able to learn a successful control strategy. Note that the reward function design is an important part for the implementation of deep RL methods in autonomous systems. In this sense, in this work a detailed reward function analysis and development was carried out to successfully satisfy the physical and operative constrains required by the AUV such as restraining the actuators sudden changes, optimization of the energy consumptions and others. In addition, an actor-critic goal-oriented architecture was developed to aid the deep agent to achieve a more generalized policy and therefore solve a bigger range of dynamic problems.

It is important to note that many previous approaches, based on deep RL framework, have used images as inputs for the state representation in order to learn a policy able to solve the control tasks. However, this type of representations are not straightforward for underwater applications where underwater images are not clear and require artificial lightning sources, which in turn increases the energy consumption of the vehicle diminishing the available mission time. In addition, the computational requirement for such an application raises the need for higher computational capability on board of the AUV, therefore increasing the energy consumption even further. Moreover, an additional image processing is needed to obtain the immediate

reward from a sequence of images, which is not a trivial problem in real-time applications. In contrast, our proposed adaptive low-level control algorithm based on deep RL framework only uses a low-level representation of the system state, based on the measures of dynamic magnitudes (linear and angular velocities), therefore higher computational costs are avoided.

The articles found in the literature with similar features to the present work, were only tested in simulation where the characterization of the systems and its environments are always available. Instead of this, our work contributes with valuable experimental results which demonstrate the capability and the successful performance of the proposed approach for AUV low-level control. During the experiments we worked with Nessie, an AUV developed at Heriot-Watt University, obtaining satisfactory results which demonstrate the feasibility of the proposed control approach to be implemented as an adaptive low-level control strategy of AUVs.

Previous works on AUVs, controlled only a limited amount of degrees of freedom, or utilized different discretization schemes to be able to control the AUV. However, this article showed that it was possible to control the six degrees-of-freedom of a real underwater vehicle by directly sending the low-level commands to the thrusters. In this sense, we think that this work is a relevant contribution for the field of autonomous underwater robotics opening a new area of research by means of including deep reinforcement learning for autonomous control formulations of AUVs. However, further research is necessary to improve the general autonomy of the robots. For example, it would be interesting to consider the possibility of enhancing our proposal by adding prior expert knowledge or combining our proposal with other low-level control techniques. Moreover, it would be also possible to include safety constraints for the training phase, or utilizing a more complex supervisor layer. It would also be interesting to test the proposed approach in other types of mobile robots due that our proposal is of a general nature and it is not only restricted to AUVs.

Our proposed approach uses recently developed ideas coming from the emergent branch of deep learning in the artificial intelligence community. Nowadays, deep reinforcement learning is at an early stage and in this paper we have contributed with real evidence for its application in robotics, particularly, for AUV applications. In this way, this fact opens a new avenue for future developments regarding deep reinforcement learning as a powerful tool for real autonomous developments in underwater robotics.

Acknowledgements

We gratefully acknowledge the support of NVIDIA Corporation with the donation of the Titan X Pascal GPU used for this research. The authors would like to especially thank Len McLean, the technician of the Heriott-Watt University, for all the help during trials, and the people of the Ocean System Laboratory for hosting this research. Particularly, we thank UNCPBA and CONICET for the financial support of Ignacio Carlucho at the Ocean System Laboratory.

References

[1] R.B. Wynn, V.A.I. Huvenne, T.P. Le Bas, B.J. Murton, D.P. Connelly, B.J. Bett, H.A. Ruhl, K.J. Morris, J. Peakall, D.R. Parsons, E.J. Sumner, S.E. Darby, R.M. Dorrell, J.E. Hunt, Autonomous Underwater Vehicles (AUVs): Their past, present and future contributions to the advancement of marine geoscience, Mar. Geol. 352 (2014) 451–468. doi:10.1016/j.margeo.2014.03.012.

[2] M. Chyba, Autonomous underwater vehicles, Ocean Eng. 36 (2009) 1. doi:10.1016/j.oceaneng.2008.12.005.

[3] A. Rozenfeld, G. Acosta, A. Sousa, H. Curti, O. Calvo, A guidance and control system proposal for autonomous pipeline inspections, Trans. Syst. Signals Devices. 5 (2010) 5–27.

[4] T.I. Fossen, Marine control systems: guidance, navigation and control of ships, rigs and underwater vehicles, Marine Cybernetics, 2002. http://www.amazon.co.uk/dp/8292356002.

[5] K. Alam, T. Ray, S.G. Anavatti, Design and construction of an autonomous underwater vehicle, Neurocomputing. 142 (2014) 16–29. doi:10.1016/j.neucom.2013.12.055.

[6] M. Knudson, K. Tumer, Adaptive navigation for autonomous robots, Rob. Auton. Syst. 59 (2011) 410–420. doi:10.1016/j.robot.2011.02.004.

[7] S.A. Gafurov, E. V. Klochkov, Autonomous Unmanned Underwater Vehicles Development Tendencies, Procedia Eng. 106 (2015) 141–148. doi:10.1016/j.proeng.2015.06.017.

[8] K.J. Åström, P.R. Kumar, Control: A perspective, Automatica. 50 (2014) 3–43. doi:10.1016/j.automatica.2013.10.012.

[9] K.J. Åström, T. Hägglund, Advanced PID control, ISA-The Instrumentation, Systems, and Automation Society, 2006.

[10] B. Jalving, The NDRE-AUV flight control system, IEEE J. Ocean. Eng. 19 (1994) 497–501. doi:10.1109/48.338385.

[11] T.I. Fossen, O.-E. Fjellstad, Robust Adaptive Control of Underwater Vehicles: A Comparative Study, IFAC Proc. Vol. 28 (1995) 66–74. doi:10.1016/S1474-6670(17)51653-5.

[12] F. Valenciaga, P.F. Puleston, O. Calvo, G.G. Acosta, Trajectory Tracking of the Cormoran AUV Based on a PI-MIMO Approach, in: Ocean. 2007 - Eur., IEEE, 2007: pp. 1–6. doi:10.1109/OCEANSE.2007.4302301.

[13] H. Sutarto, A. Budiyono, Development of linear parameter varying control system for autonomous underwater vehicle, Indian J. Geo-Marine Sci. 40 (2011) 275–286.

[14] P. Sarhadi, A.R. Noei, A. Khosravi, Model reference adaptive PID control with anti-windup compensator for an autonomous underwater vehicle, Rob. Auton. Syst. 83 (2016) 87–93. doi:10.1016/j.robot.2016.05.016.

[15] S. Galeani, S. Tarbouriech, M. Turner, L. Zaccarian, A Tutorial on Modern Anti-windup Design, Eur. J. Control. 15 (2009) 418–440. doi:10.3166/ejc.15.418-440.

[16] A.H. Tahoun, Anti-windup adaptive PID control design for a class of uncertain chaotic systems with input saturation, ISA Trans. 66 (2017) 176–184. doi:10.1016/j.isatra.2016.10.002.

[17] P. Sarhadi, A.R. Noei, A. Khosravi, Model reference adaptive autopilot with anti-windup compensator for an autonomous underwater vehicle: Design and hardware in the loop implementation results, Appl. Ocean Res. 62 (2017) 27–36. doi:10.1016/j.apor.2016.11.005.

[18] B. Ferreira, M. Pinto, A. Matos, N. Cruz, F. Deec, Control of the MARES Autonomous Underwater Vehicle, in: Ocean. 2009, MTS/IEEE Biloxi, 2009. doi:10.23919/OCEANS.2009.5422133.

[19] L. Lapierre, B. Jouvencel, Robust Nonlinear Path-Following Control of an AUV, IEEE J. Ocean. Eng. 33 (2008) 89–102. doi:10.1109/JOE.2008.923554.

[20] S.A. Wadoo, S. Sapkota, K. Chagachagere, Optimal control of an autonomous underwater vehicle, in: 2012 IEEE Long Isl. Syst. Appl. Technol. Conf., IEEE, 2012: pp. 1–6. doi:10.1109/LISAT.2012.6223100.

[21] B. Geranmehr, S.R. Nekoo, Nonlinear suboptimal control of fully coupled non-affine six-DOF autonomous underwater vehicle using the state-dependent Riccati equation, Ocean Eng. 96 (2015) 248–257. doi:10.1016/j.oceaneng.2014.12.032.

[22] N. Fischer, D. Hughes, P. Walters, E.M. Schwartz, S. Member, W.E. Dixon, S. Member, Nonlinear RISE-Based Control of an Autonomous Underwater Vehicle, IEEE Trans. Robot. 30 (2014) 845–852. doi:10.1109/TRO.2014.2305791.

[23] N. Sadegh, R. Horowitz, Stability and Robustness Analysis of a Class of Adaptive Controllers for Robotic Manipulators, Int. J. Rob. Res. 9 (1990) 74–92. doi:10.1177/027836499000900305.

[24] J.-E. Slotine, M.D. Di Benedetto, Hamiltonian adaptive control of spacecraft, IEEE Trans. Automat. Contr. 35 (1990) 848–852. doi:10.1109/9.57028.

[25] G. Antonelli, On the use of adaptive/integral actions for six-degrees-of-freedom control of autonomous underwater vehicles, IEEE J. Ocean. Eng. 32 (2007) 300–312. doi:10.1109/JOE.2007.893685.

[26] G. Antonelli, F. Caccavale, S. Chiaverini, G. Fusco, A Novel Adaptive Control Law for Underwater Vehicles, 11 (2003) 221–232.

[27] G. Antonelli, S. Chiaverini, N. Sarkar, M. West, Adaptive control of an autonomous underwater vehicle: experimental results on ODIN, IEEE Trans. Control Syst. Technol. 9 (2001) 756–765. doi:10.1109/87.944470.

[28] C. Barbalata, V. De Carolis, M.W. Dunnigan, Y. Petillot, D. Lane, An adaptive controller for autonomous underwater vehicles, in: 2015 IEEE/RSJ Int. Conf. Intell. Robot. Syst., IEEE, Hamburg, 2015: pp. 1658–1663. doi:10.1109/IROS.2015.7353590.

[29] R. Rout, B. Subudhi, Inverse optimal self-tuning PID control design for an autonomous underwater vehicle, Int. J. Syst. Sci. 48 (2017) 367–375. doi:10.1080/00207721.2016.1186238.

[30] S. Chen, S.A. Billings, Representations of non-linear systems: the NARMAX model, Int. J. Control. 49 (1989) 1013–1032. doi:10.1080/00207178908559683.

[31] M. Narasimhan, S.N. Singh, Adaptive optimal control of an autonomous underwater vehicle in the dive plane using dorsal fins, 33 (2006) 404–416. doi:10.1016/j.oceaneng.2005.04.017.

[32] S. Haykin, Neural Networks: A Comprehensive Foundation, 2nd ed., Prentice Hall, 1998. http://www.amazon.com/dp/0132733501.

[33] P.W.J. van de Ven, C. Flanagan, D. Toal, Neural network control of underwater vehicles, Eng. Appl. Artif. Intell. 18 (2005) 533–547. doi:10.1016/j.engappai.2004.12.004.

[34] Y. Shi, W. Qian, W. Yan, J. Li, Adaptive Depth Control for Autonomous Underwater Vehicles Based on Feedforward Neural Networks, Intell. Control Autom. 4 (2007) 207–218. doi:10.1007/978-3-540-37256-1_29.

[35] D. Zhu, B. Sun, The bio-inspired model based hybrid sliding-mode tracking control for unmanned underwater vehicles, Eng. Appl. Artif. Intell. 26 (2013) 2260–2269. doi:10.1016/j.engappai.2013.08.017.

[36] L.A. Zadeh, Outline of a New Approach to the Analysis of Complex Systems and Decision Processes, IEEE Trans. Syst. Man. Cybern. SMC-3 (1973) 28–44. doi:10.1109/TSMC.1973.5408575.

[37] S.M. Smith, G.J.S. Rae, D.T. Anderson, A.M. Shein, Fuzzy Logic Control of an Autonomous Underwater Vehicle, Control Eng. Pract. 2 (1994) 321–331. doi:10.1016/0967-0661(94)90214-3.

[38] P.A. DeBitetto, Fuzzy logic for depth control of unmanned undersea vehicles, in: Proc. IEEE Symp. Auton. Underw. Veh. Technol., IEEE, 1995: pp. 233–241. doi:10.1109/AUV.1994.518630.

[39] J. Guo, F.-C. Chiu, C.-C. Huang, Design of a sliding mode fuzzy controller for the guidance and control of an autonomous underwater vehicle, Ocean Eng. 30 (2003) 2137–2155. doi:10.1016/S0029-8018(03)00048-9.

[40] S.M. Smith, G.J.S. Rae, D.T. Anderson, Applications of fuzzy logic to the control of an autonomous underwater vehicle, in: [Proceedings 1993] Second IEEE Int. Conf. Fuzzy Syst., IEEE, 1993: pp. 1099–1106. doi:10.1109/FUZZY.1993.327361.

[41] L.A. Zadeh, Fuzzy logic, neural networks, and soft computing, Commun. ACM. 37 (1994) 77–84. doi:10.1145/175247.175255.

[42] B. Raeisy, A.A. Safavi, A.R. Khayatian, Optimized fuzzy control design of an autonomous underwater vehicle, Iran. J. Fuzzy Syst. 9 (2012) 25–41.

[43] M.H. Khodayari, S. Balochian, Modeling and control of autonomous underwater vehicle (AUV) in heading and depth attitude via self-adaptive fuzzy PID controller, J. Mar. Sci. Technol. 20 (2015) 559–578. doi:10.1007/s00773-015-0312-7.

[44] P.S. Londhe, S. Mohan, B.M. Patre, L.M. Waghmare, Robust task-space control of an autonomous underwater vehicle-manipulator system by PID-like fuzzy control scheme with disturbance estimator, Ocean Eng. 139 (2017) 1–13.

doi:10.1016/J.OCEANENG.2017.04.030.

[45] H.N. Esfahani, V. Azimirad, M. Danesh, A Time Delay Controller included terminal sliding mode and fuzzy gain tuning for Underwater Vehicle-Manipulator Systems, Ocean Eng. 107 (2015) 97–107. doi:10.1016/J.OCEANENG.2015.07.043.

[46] R.S. Sutton, A.G. Barto, Reinforcement learning: An introduction, MIT Press, 1998. http://books.google.com/books?id=CAFR6IBF4xYC.

[47] J. Kober, J.A. Bagnell, J. Peters, Reinforcement Learning in Robotics: A Survey, Int. J. Rob. Res. (2013) 579–610. http://link.springer.com/chapter/10.1007/978-3-642-27645-3_18 (accessed August 8, 2015).

[48] D.R. Parhi, S. Kundu, Review on Guidance , Control and Navigation of Autonomous Underwater Mobile Robot, Int. J. Artif. Intell. Comput. Res. 4 (2012) 21–31.

[49] M.A.-R. Mohammad Abdel Kareem Jaradat, Reinforcement based mobile robot navigation in dynamic environment. Robotics Comput-Integr Manuf, Robot. Comput. Integr. Manuf. 27 (2011) 135–149. doi:10.1016/j.rcim.2010.06.019.

[50] M. Deisenroth, C.E. Rasmussen, Efficient Reinforcement Learning for Motor Control, in: 2009. http://eprints.pascal-network.org/archive/00005478/ (accessed May 16, 2011).

[51] P. Dayan, K.C. Berridge, Model-Based and Model-Free Pavlovian Reward Learning: Revaluation, Revision and Revelation, Cogn Affect Behav Neurosci. 14 (2014) 473–492. doi:10.1086/498510.

[52] Y. Chebotar, K. Hausman, M. Zhang, G. Sukhatme, S. Schaal, S. Levine, Combining Model-Based and Model-Free Updates for Trajectory-Centric Reinforcement Learning, (2017). http://arxiv.org/abs/1703.03078 (accessed September 19, 2017).

[53] C.J.C.H. Watkins, P. Dayan, Q-learning, Mach. Learn. 8 (1992) 279–292. doi:10.1007/BF00992698.

[54] T. Hester, M. Quinlan, P. Stone, RTMBA: A real-time model-based reinforcement learning architecture for robot control, in: 2012: pp. 85–90. doi:10.1109/ICRA.2012.6225072.

[55] C. Gaskett, D. Wettergreen, A. Zelinsky, Reinforcement Learning applied to the control of an Autonomous Underwater Vehicle, in: 1999: pp. 125–131. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.33.8469&rep=rep1&type=pdf.

[56] M. Carreras, J. Yuh, J. Batlle, P. Ridao, A behavior-based scheme using reinforcement learning for autonomous underwater vehicles, IEEE J. Ocean. Eng. 30 (2005) 416–427. doi:10.1109/JOE.2004.835805.

[57] A. El-Fakdi, M. Carreras, N. Palomeras, P. Ridao, Autonomous underwater vehicle control using reinforcement learning policy search methods, in: 2005: p. 793–798 Vol. 2. doi:10.1109/OCEANSE.2005.1513157.

[58] A. El-Fakdi, Gradient-based reinforcement learning techniques for underwater robotics behavior learning, 2010. http://tesisenred.net/handle/10803/7610 (accessed April 26, 2012).

[59] A. El-Fakdi, M. Carreras, Two-step gradient-based reinforcement learning for underwater robotics behavior learning, Rob. Auton. Syst. 61 (2013) 271–282. doi:10.1016/j.robot.2012.11.009.

[60] K. Blekas, K. Vlachos, RL-based path planning for an over-actuated floating vehicle under disturbances, Rob. Auton. Syst. 101 (2018) 93–102. doi:S0921889017301884.

[61] G. Frost, D.M. Lane, Evaluation of Q-learning for search and inspect missions using underwater vehicles, in: 2014 Ocean. - St. John's, IEEE, 2014: pp. 1–6. doi:10.1109/OCEANS.2014.7003088.

[62] G. Frost, F. Maurelli, D.M. Lane, Reinforcement learning in a behaviour-based control architecture for marine archaeology, in: Ocean. 2015 - Genova, IEEE, 2015: pp. 1–5. doi:10.1109/OCEANS-Genova.2015.7271619.

[63] R. Cui, C. Yang, Y. Li, S. Sharma, Adaptive Neural Network Control of AUVs With Control Input Nonlinearities Using Reinforcement Learning, IEEE Trans. Syst. Man, Cybern. Syst. 47 (2017) 1019–1029. doi:10.1109/TSMC.2016.2645699.

[64] M. Riedmiller, Neural Fitted Q Iteration – First Experiences with a Data Efficient Neural Reinforcement Learning Method, in: Springer, Berlin, Heidelberg, 2005: pp. 317–328. doi:10.1007/11564096_32.

[65] T. Degris, P.M. Pilarski, R.S. Sutton, Model-Free reinforcement learning with continuous action in practice, in: 2012 Am. Control Conf., IEEE, 2012: pp. 2177–2182. doi:10.1109/ACC.2012.6315022.

[66] Y. LeCun, Y. Bengio, G. Hinton, Deep learning, Nature. 521 (2015) 436–444. doi:10.1038/nature14539.

[67] A. Krizhevsky, I. Sutskever, G.E. Hinton, ImageNet classification with deep convolutional neural networks, Proceeding NIPS'12 Proc. 25th Int. Conf. Neural Inf. Process. Syst. 25 (2012) 1097–1105.

http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.299.205 (accessed September 22, 2017).

[68] V. Mnih, K. Kavukcuoglu, D. Silver, A.A. Rusu, J. Veness, M.G. Bellemare, A. Graves, M. Riedmiller, A.K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, D. Hassabis, Human-level control through deep reinforcement learning, Nature. 518 (2015) 529–533. doi:10.1038/nature14236.

[69] T.P. Lillicrap, J.J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, D. Wierstra, Continuous control with deep reinforcement learning, in: ICLR 2016, 2016: pp. 1–14. doi:10.1561/2200000006.

[70] D. Silver, N. Heess, T. Degris, D. Wierstra, M. Riedmiller, Deterministic Policy Gradient Algorithms, in: 31 St Int. Conf. Mach. Learn., 2014. http://proceedings.mlr.press/v32/silver14.pdf (accessed September 22, 2017).

[71] S. Ioffe, C. Szegedy, Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift, in: 32 Nd Int. Conf. Mach. Learn., 2015. doi:10.1007/s13398-014-0173-7.2.

[72] A. Yu, R. Palefsky-Smith, R. Bedi, Deep Reinforcement Learning for Simulated Autonomous Vehicle Control, 2016.

[73] A. Ganesh, J. Charalel, M. Das Sarma, N. Xu, Deep Reinforcement Learning for Simulated Autonomous Driving, 2016. http://cs229.stanford.edu/proj2016/report/Ganesh-Charalel-DasSarma-Xu-DeepReinforcementLearningForSimulatedAutonomousDriving-report.pdf (accessed September 21, 2017).

[74] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G.S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mane, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viegas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, X. Zheng, TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems, (2015). http://arxiv.org/abs/1603.04467 (accessed September 21, 2017).

[75] F. Chollet, Keras: Deep learning library for Theano and TensorFlow, Https://keras.io/. (2016) https://keras.io/. https://keras.io/.

[76] D. Loiacono, P.L. Lanzi, J. Togelius, E. Onieva, D.A. Pelta, M. V Butz, T.D. Lönneker, L. Cardamone, D. Perez, Y. Sáez, M. Preuss, J. Quadflieg, The 2009 Simulated Car Racing Championship, IEEE Trans. Comput. Intell. AI Games. 2 (2010) 131–147. doi:10.1109/TCIAIG.2010.2050590.

[77] A. El Sallab, M. Abdou, E. Perot, S. Yogamani, Deep Reinforcement Learning framework for Autonomous Driving, Soc. Imaging Sci. Technol. 7 (2017) 70–76. doi:10.2352/ISSN.2470-1173.2017.19.AVM-023.

[78] R. Yu, Z. Shi, C. Huang, T. Li, Q. Ma, Deep Reinforcement Learning Based Optimal Trajectory Tracking Control of Autonomous Underwater Vehicle, in: 36th Chinese Control Conf., 2017: pp. 4958–4965.

[79] D.P. Kingma, J. Ba, Adam: A Method for Stochastic Optimization, in: 3rd Int. Conf. Learn. Represent., San Diego, 2015. http://arxiv.org/abs/1412.6980 (accessed September 21, 2017).

[80] T. Serre, G. Kreiman, M. Kouh, C. Cadieu, U. Knoblich, T. Poggio, A quantitative theory of immediate visual recognition, in: Prog. Brain Res., 2007: pp. 33–56. doi:10.1016/S0079-6123(06)65004-8.

[81] I. Goodfellow, Y. Bengio, A. Courville, Deep learning, The MIT Press, London, 2017.

[82] J.T. Springenberg, A. Dosovitskiy, T. Brox, M. Riedmiller, Striving for Simplicity: The All Convolutional Net, ICLR 2015. (2015). http://arxiv.org/abs/1412.6806 (accessed October 10, 2017).

[83] Xiaoheng Jiang, Yanwei Pang, Xuelong Li, J. Pan, Y. Xie, Deep neural networks with Elastic Rectified Linear Units for object recognition, Neurocomputing. (2017). doi:10.1016/J.NEUCOM.2017.09.056.

[84] R. Arora, A. Basu, P. Mianjy, A. Mukherjee, Understanding Deep Neural Networks with Rectified Linear Units, 2017. http://arxiv.org/abs/1611.01491.

[85] S. Lange, T. Gabel, M. Riedmiller, Batch Reinforcement LEarning, 2012. doi:10.1007/978-3-642-27645-3.

[86] N. Valeyrie, F. Maurelli, P. Patron, J. Cartwright, B. Davis, Y. Petillot, Nessie v turbo: a new hover and power slide capable torpedo shaped auv for survey, inspection and intervention, in: AUVSI North Am. 2010 Conf., 2010.

[87] C. Barbalata, M.W. Dunnigan, Y. Petillot, Dynamic coupling and control issues for a lightweight underwater vehicle manipulator system, in: 2014 Ocean. - St. John's, IEEE, 2014: pp. 1–6. doi:10.1109/OCEANS.2014.7002989.

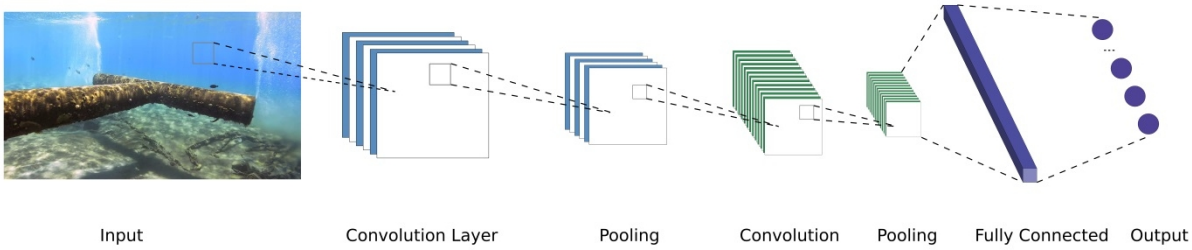


Figure 1. General deep neural network architecture.

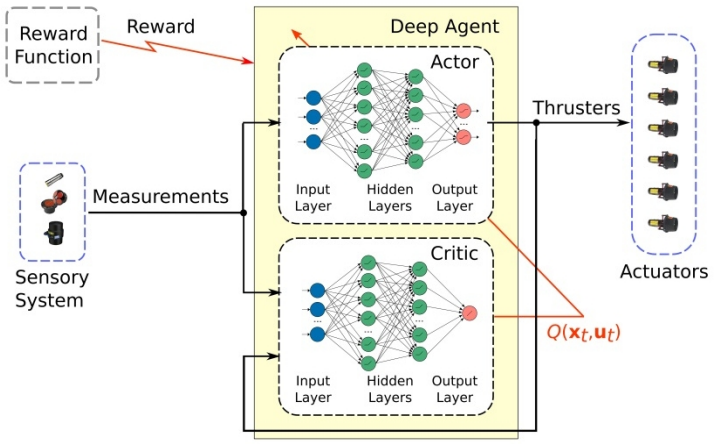


Figure 2. Actor-critic architecture using deep fully connected neural network of ReLU layers as function approximators.

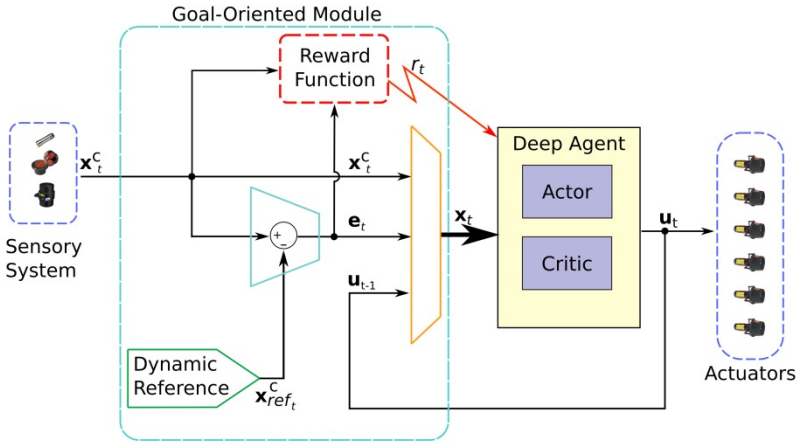
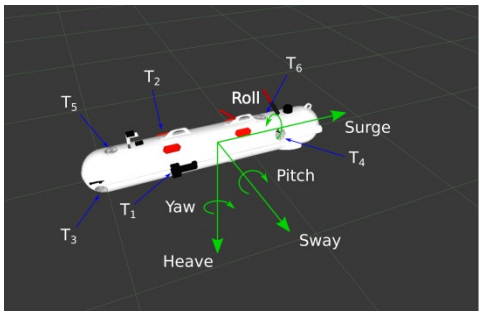


Figure 3. Goal-oriented control architecture based on actor-critic architecture.

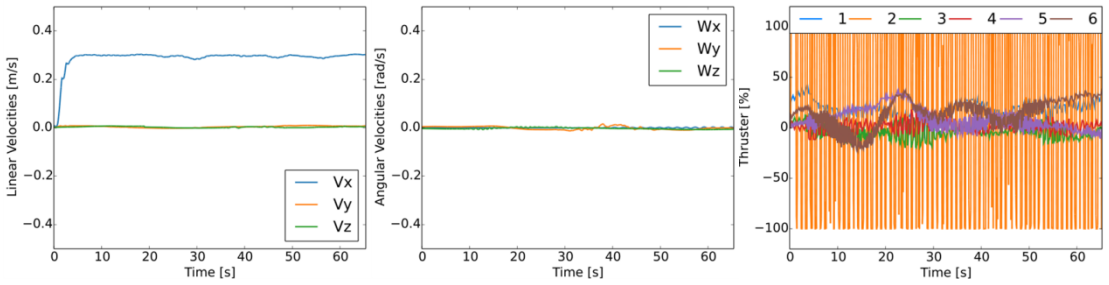
(a)



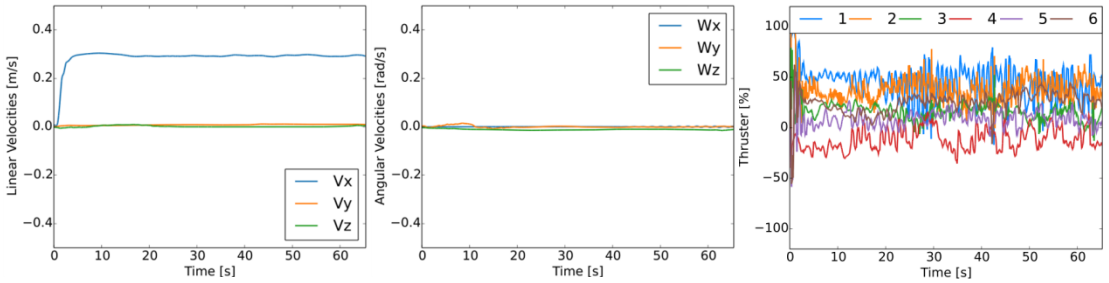
(b)

Figure 4. The autonomous underwater vehicle, Nessie VII, designed and built at Heriot-Watt University.

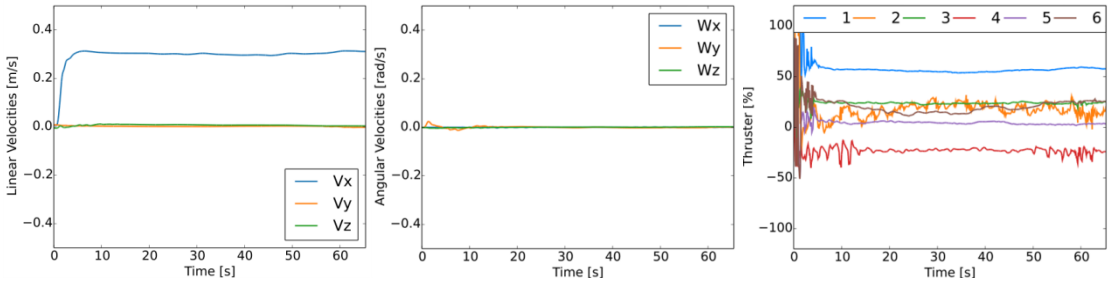
a) Experimental platform; **b**) Body reference system.



(a)



(b)



(c)

Figure 5. Comparative essays for different structures for the reward function.

a) Results for a reward function of Eq. **Error! Reference source not found.** with $\lambda \neq 0$ and $\zeta = \xi = 0$; **b**) Results for a reward function of Eq. **Error! Reference source not found.** with $\lambda \neq 0$, $\zeta \neq 0$ and $\xi = 0$; **c**) Results for a reward function of Eq. **Error! Reference source not found.** with $\lambda \neq 0$, $\zeta \neq 0$ and $\xi \neq 0$.

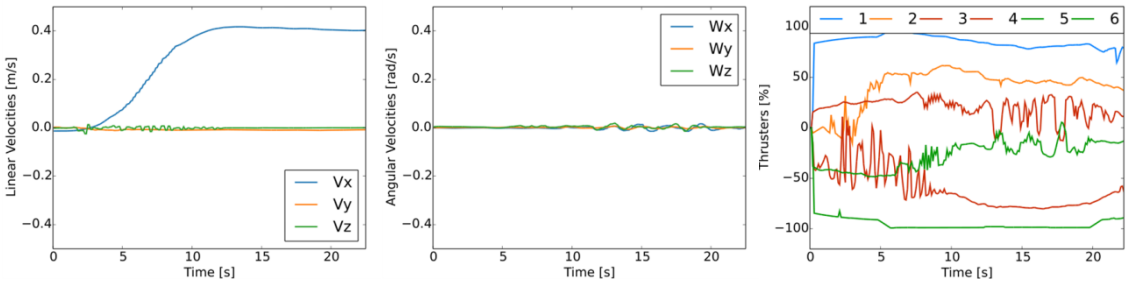


Figure 6. Results of a training episode for $\mathbf{x}_{ref}^{c} = (0.4\ \mathrm{m/s}, 0, 0, 0, 0)$ during the execution of Algorithm 1.

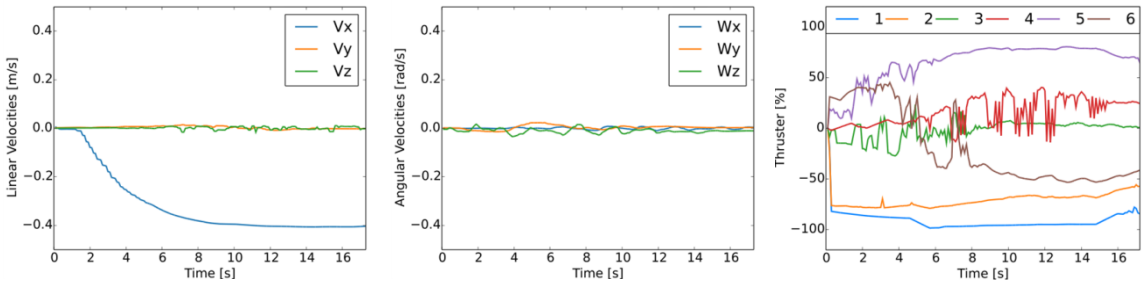


Figure 7. Results of a training episode for $\mathbf{x}_{ref}^{c} = (-0.4\ \mathrm{m/s}, 0, 0, 0, 0)$ during the execution of Algorithm 1.

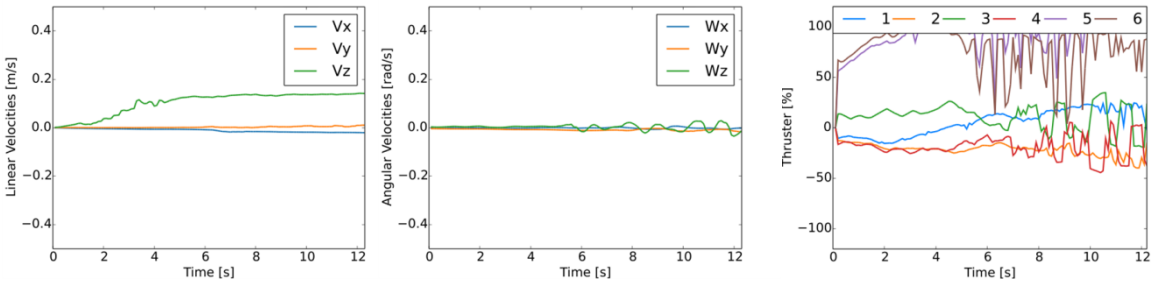


Figure 8. Results of a training episode for $\mathbf{x}_{ref}^{c} = (0{,}1,\ 0.18\ \mathrm{m/s},\ 0,\ 0)$ during the execution of Algorithm 1.

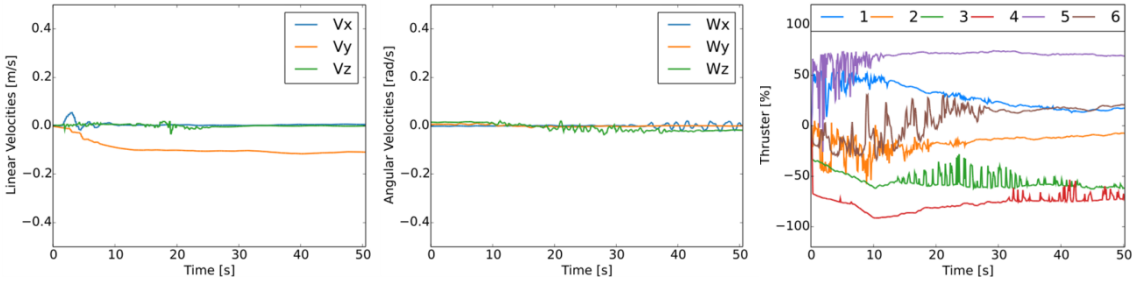


Figure 9. Results of a training episode for $\mathbf{x}_{r\acute{e}f}^{\mathrm{c}} = (0, -0.1\mathrm{m/s}, 0, 0, 0)$ during the execution of Algorithm 1.

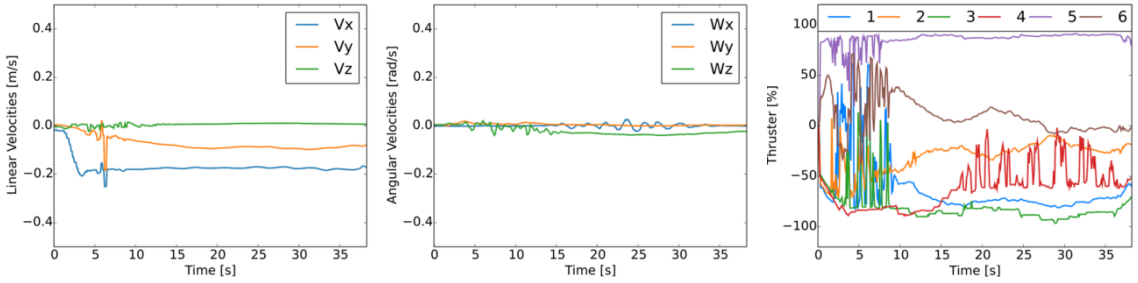


Figure 10. Results of a training episode for $\mathbf{x}_{ref}^{c} = (-0.2, -0.1\mathrm{m/s}, 0, 0, 0)$ during the execution of Algorithm 1.